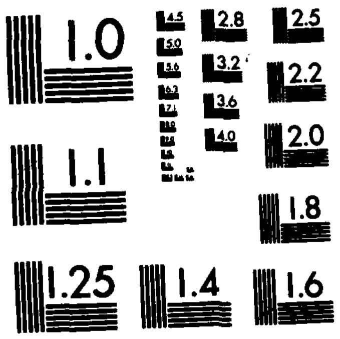
AD-A124 661 AN INTERACTIVE BOMBING MISSION SIMULATION WITH COMPUTER GRAPHICS INTERFACE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M J GOCI 1/2

UNCLASSIFIED DEC 82 AFIT/GCS/MA/82D-4 F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

(1)

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/MA/82D-4

AN INTERACTIVE
BOMBING MISSION SIMULATION
WITH COMPUTER GRAPHICS INTERFACE

THESIS

AFIT/GCS/MA/82D-4

MICHAEL J. UCCI
CAPT USAF

AFIT/GCS/EE/82D-4

AN INTERACTIVE BOMBING MISSION SIMULATION WITH COMPUTER GRAPHICS INTERFACE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Michael J. Goci, B.S.
Capt USAF

Graduate Computer Systems
December 1982

Approved for public release; distribution unlimited.

PREFACE

This report is the result of my efforts to design, implement, and test an interactive flight simulation with graphics interface for a bombing mission program used by analysts in the Avionics Laboratory, Air Force Wright Aeronautical Laboratories. The document is written for readers with some knowledge of computer programming in high level languages and interactive computer graphics. The design and analysis is documented using the SADT approach. The software is written in FORTRAN, and the graphics are accomplished using PLOT-10 software on a TEKTRONIX 4016 terminal.

I wish to express my appreciation to Professor Charles W. Richard for his support and guidance throughout the development of this project. Thanks also to Mr. William K. McQuay of the Avionics Laboratory for offering the thesis topic, then following through on its development and implementation. Finally, I wish to thank my wife, Ann, for her support and understanding throughout this project, and in particular for typing this thesis.

Michael J. Goci

CONTENTS

CONTENTS (cont'd)

LIST OF FIGURES

LIST OF FIGURES (cont'd.)

AFIT/GCS/MA/82D-4

ABSTRACT

An interactive flight simulation with computer graphics interface was designed using top-down structured analysis techniques. The project converts a passive bombing mission simulation used in the Avionics Laboratory, Air Force Wright Aeronautical Laboratories at Wright-Patterson AFB, into an interactive, real-time, man-in-loop simulation. The design was documented using SofTech's Structured Analysis and Design Technique (SADT) then coded in FORTRAN. The graphics were implemented using TEKTRONIX PLOT-10 software and the system operates on a VAX-11/780 computer coupled through a TEKTRONIX 4016 terminal.

AN INTERACTIVE BOMBING MISSION SIMULATION WITH COMPUTER GRAPHICS INTERFACE

I <u>Introduction</u>

A nation's success or failure in a conventional war is ultimately dependent on that nation attaining and maintaining air superiority. To attain air superiority requires the capability to effectively penetrate the enemy defensive system with tactical and strategic weapons. Manned bombers comprise a large percentage of both tactical and strategic weapons. Whether a bomber will effectively penetrate an enemy's defensive network and destroy its planned targets is determined by a combination and interaction of several factors. The first, and most critical factor is the type, number, and planned use of electronic countermeasure equipment. Next are the aircraft parameters of airspeed, altitude, and approach heading relative to the defensive system. Since intelligence data is seldom perfectly accurate, and what was true of the enemy system yesterday may have been changed by today anyway, another major consideration is the pilot's skill, intuition and reaction to the hostile environment as the mission develops. Uncontrollable elements make up a fourth category. Weather, such as thunderstorms or heavy precipitation, is notorious for producing unusual returns on radar. These effects may also hide radar returns which normally would have been observed. Chance also plays an untold role. A jammer might indicate on in the cockpit, but actually not produce a signal, or an enemy SAM system may not fire due to some malfunction.

Clearly, with this many factors to consider, and with the innumerable potential combinations and interactions among these factors, it is a complicated problem to determine the overall effect of a change in one, or any combination of the input parameters. It obviously is not cost effective, and in many cases not even possible to do inflight testing of each parameter adjustment in order to optimize the mission profile. The Analysis Group attacked this problem by using several different simulation programs to analyze the effectiveness of various electronic countermeasure systems in use by the U. S. Air Force.

The mission simulation programs, called the Avionics Air Defense Evaluation Model (AADEM), currently used by the Analysis Group are written mostly in FORTRAN IV, with some routines written in updated versions of FORTRAN. AADEM is run on a Digital Equipment Corporation VAX 11/780 Computer System. The programs require that the aircraft flight path be defined completely before program execution. This requires that all aircraft events such as heading, airspeed, or altitude changes, electronic countermeasure (ECM) configuration changes, and target selection/weapon firing must be predetermined and formatted as input to the various simulation programs. An error in this input data results in an invalid simulation run, which may require several days to locate and correct the error, regenerate the input deck, and rerun. Also, in order to make only one small change in the mission profile requires that the entire input deck be reenterred with the modification.

Problem Description

Because of the above limitations, the Avionics Laboratory requires a flight simulation of an aircraft penetrating an enemy deployment of surface-to-air missiles (SAM) and anti-aircraft artillery (AAA) with the capability to simulate interactive "pilot" actions. Ideally, the program should process an initial predefined flight path using the same routines currently in use by the Analysis Group. In addition, it should give an interactive "pilot" the information normally available in an aircraft cockpit, in a graphical format, and allow for modification to the mission profile in as close to actual time as possible. [Ref 1]

With this type of simulation program, the analysts will be able to produce a test environment which parallels an actual flight much more closely than the passive simulations currently in use. Also, any errors in the predefined flight path can be corrected by the "pilot" during simulation execution, thereby reducing the amount of time needed to obtain a valid test run.

Thesis Objectives/Approach

The approach was governed by two conflicting ideals: We wanted to implement and test a model rather than merely design one. However, to build a complete bombing mission simulation is too large a project for a thesis effort. The compromise was to be satisfied with only partial development of some aspects of the simulation. The particular details of these compromises follow.

Using software engineering principles and techniques, the simulation was structured into modules, with each detail in its own module. In this way the modules can be enhanced later to whatever degree is desired. In addition to the modeling of a bombing mission, this project uses TEKTRONIX PLOT-10 [Ref 2] graphics to interface with the "pilot" and thereby furnish the information normally available in an aircraft cockpit. PLOT-10 rather than Core Standard [Ref 3] graphics were used not by choice, but because no Core Standard package was available on the Avionics Laboratory computer when this project was begun. Modular design with a clean graphics interface will make translation to other graphics routines straight forward.

As partial justification for my approach to this project I offer the folowing by Robert Shannon, author of many books and articles dealing with computer modeling.

> The approach to the successful building of models appears to proceed on the basis of elaboration and enrichment. One begins with a very simple model and attempts to move in an evolutionary fashion toward a more elaborate model

> that reflects the complex situation more clearly. Analogy or association with previously well-developed structures appears to play an important role in determining the starting point for this process of elaboration and enrichment. The process of elaboration and enrichment involves a constant interaction and feedback process between the real world situation and the model. There is a constant interplay between the modification of the model and a confrontation with the data generated. As each version of the model is tested and attempts to validate it are made, a new version is produced that leads to a retesting and revalidation. As long as the model is computationally tractable, the analyst may seek further enrichment or complication of the assumptions. When it becomes intractable or cannot be solved, he resorts to simplification and further abstraction.
> [Ref 4:19,20]

Some of the design work of the interactive graphics for this project was done by Capt. Randy Krause as his AFIT thesis in 1978. [Ref 5] Since Capt. Krause's design was to be implemented as a time-shared job on the ASD CYBER Computer using a CDC CYBER GRAPHICS terminal, several modifications were made to his design. His work did serve as background for this project, and as a reference against which to cross check ideas.

In the next chapter, the fundamentals of simulation and modeling, which served as the cornerstone of my project, are briefly discussed. This chapter can be skipped with no loss of continuity.

II Simulation and Modeling

The fundamental concept of systems analysis is that changing one aspect of a system might very well change, or create the need for changes in other related, or seemingly unrelated parts of that system. As man-made systems become increasingly large, and the interrelationships of the subparts become increasingly complex, engineers and planners have a more and more difficult time predicting and controlling the outcomes of various inputs. One very useful tool in helping to observe the outcome of various inputs, and thereby improve the understanding of a complicated system, is computer simulation.

Simulation is the process of designing a model of a real system and conducting experiments with this model either to better understand the behavior of the system or to evaluate various strategies (within the limits imposed by a set of criteria) for the operation of the system. To simulate, then, is to both construct a model which (hopefully) accurately describes the interrelationships of all parts of a system, and then to use this model to predict the effects that will be produced by changes in either the system itself, or some of the methods of its operation.

Three methods of studying a bombing mission are typically employed: digital models, simulators, and flight tests. A digital model is another name for a computer model which usually has all parameters and events established prior to beginning the simulation. A simulator is a computer model which has a man-in-loop facility to incorporate dynamic inputs and changes as the simulation progresses. It uses hardware and realistic displays of actual equipment to simulate the real world system. Flight tests involve using real aircraft with simulated threats and defense systems with associated operations performed in a controlled environment. In choosing which method to employ the analyst must evaluate several advantages and limitations.

It's easy to see that digital models and simulators offer much stricter control than can be achieved with flight testing and it is much easier to make configuration changes. Computer simulation is also considerably less costly and requires less instrumentation to collect output data. Simulators are an obvious improvement over purely digital models since the human interactions with realistic simulations of ECM hardware and radar equipment enhance realism. But they still are limited compared to flight tests since there are restrictions on the number of aircraft, ECM capabilities, and real world interactions which can be simulated without dedicating more computer power than is typically available. However, these limits are rapidly melting away with advances in hardware.

Flight tests can do anything that a simulation can, if the analyst will support the overhead costs required to establish the mission profile. In addition, they can evaluate equipment, tactics, and operational readiness. The advantages include the capability to use multiple aircraft, multiple penetration aids, incorporate more real world interactions, and thus a higher degree of realism. Its limitations include tremendous costs in personnel and instrumentation to collect data, limited control, fixed geography for the test flight, and restrictions on the ability to alter the defensive structure. Figure 1 below summarizes the three methods of collecting system data according to several measures of value.

	DIGITAL MODELS	SIMULATORS	FLIGHT TESTS	REAL WORLD
REALISM	Medium/low	Good	High	Highest
CONTROL	Easy	Easy	Difficult	Not Possible
INSTRUMENT COSTS	Low	Moderate	High	High
DATA REDUCTION COSTS	Low	Moderate	High	High
OVERALL COSTS	Low	Moderate	High	Very High
TIME REQUIREMENTS	Days	Weeks	Months	Year
VERIFICATION SOURCE	Simulators Flight test Real World	Flight test Real World	Real World	

[Ref 6:172]

FIGURE 1
A Comparsion of Simulations with Real World

An additional advantage of simulation is that it forces the model builder to thoroughly analyze the system and study the various interrelations of the parts. Manipulating the various subsystems while building the model often educates the builder and gives him or her more and different ideas of potential changes to the inputs to improve system performance. There is one more disadvantage to simulation which is often addressed. As stated above, the outputs of a simulation merely result from a combination of the inputs, and the model builder's assumptions about the real world. The outputs are always imprecise, but it is difficult, if not impossible to determine how imprecise. Since the outputs are usually numbers, the number of digits printed may bear little relation to the accuracy of the model's assumptions. If the printing of the output is not carefully formatted, the results can be very misleading to analysts unfamiliar with the capabilities of the model.

Clearly, there are many potential advantages and disadvantages in using a computer simulation rather than direct experimentation with the real system. Obviously, direct experimentation precludes the problem of building a model which is not an accurate representation of the real world, and from it deriving false predictions of the future. However, sometimes, as in the case of a bombing mission, direct experimentation is not possible. In fact, Shannon gives six conditions under which an analyst should consider the use of simulation. Of the six, the following three fit this problem.

1) A complete mathematical formulation of the problem does not exist or analytical methods of solving the mathematical model have not yet been developed. Many waiting line (queueing) models are in this category.
2) It is desired to observe a simulated history of the process over a period of time in addition to estimating certain parameters.
3) Simulation may be required for systems or processes with long time frames. Simulation affords complete control over time, since a phenomenon may be speeded up or slowed down at will. Analysis of urban decay problems is in this category.

[Ref 4:11]

The idea of a simulation to solve this problem is initially appealing, and appears appropriate in that it fits half of Shannon's criteria as stated above, but to keep it in perspective remember that a simulation never solves a problem. A simulation is run with a given set of inputs, and from the inputs, and the criteria of the model, it generates outputs. It is incapable of reaching a conclusion on its own, but serves only as a tool to help the analyst understand the behavior of the system and the affects of various changes of inputs.

Now, if we agree that a computer model is a valid tool with which to attack a problem, how do we go about building one? Clearly we must analyze the problem and try to extract the essential features without leaving out any critical details. Next, we structure our set of essential features into an approximation (possibly very crude) of the real system. We then enrich and elaborate this approximation until a useful predictor results.

Several sources list the following set of guidelines to help an analyst design a model.

1) Factor the system into a collection of smaller subsystems.
2) Clearly state the objectives of the simulation.
3) Seek analogies to other models or systems.
4) Consider specific instances of the problem.
5) Establish some symbols.
6) Write down the obvious.
7) If a usable model results, enrich it. If not, simplify.

[Ref 7]

Simplification is the key which opens any system to the model builder. For example, make variables into constants, restrict the boundaries of the system and use linear approximations. These approximations can be enriched and the model improved after the first model is operating. The bottom line is don't get bogged down with details too early in the project. Clearly, the standard approach to model building is not to try one all-encompassing simulation on the first attempt, but rather to begin with a very basic and straight forward model, and then enrich it as the system becomes increasingly familiar to the builder and user. Since model building is inherently an evolutionary process, a computer model, even more than other computer software must be written modularly, with flexibility and probability of change foremost in the programmer's mind.

Considering that models, by nature, evolve, a "good" model is:

1) Simple for the user, to understand, control, and communicate with
2) Robust, in that it does not give absurd answers
3) Complete on important issues
4) Adaptive, and modular to allow for enrichment
5) Evolutionary, in that it should start simply and become more complex, in conjunction with the user

It is not my purpose in this report to thoroughly discuss all factors which need to be considered while designing and building a computer model. I merely wanted to touch on what I consider the most important considerations, and to somewhat justify my approach to this interactive simulation effort. For those readers interested in more information on computer simulation and modeling, I highly recommend references 4, 6, 7, and 8 from my bibliography.

In closing this chapter, I present this "summary" about model building:

> There is no hard and fast rule about how the problem is originally formatted, i.e., how one looks at it in the first place. There are no magic formulas for deciding what should be included in the model in the form of variables and parameters, descriptive relationships and constraints, and criterion for judgement of effectiveness. Remember that nobody solves the problem; rather, everybody

> solves the model that he has constructed of the problem. This concept helps to keep the model and the art of modeling in the proper perspective.
> [Ref 4:21]

With these words of guidance, I am ready to establish the project specifications.

III System Specifications

Development of this system, like any software enginering effort, must begin by laying down a conceptual framework for the project. Normally, the final user of a software package will designate in considerable detail the requirements and capabilities for the proposed system. However, for a thesis effort, it is more common for the project sponsor to state desires in more general terms, and leave the requirements definition and project scoping to the graduate student and the thesis advisor. This chapter will document the desired capabilities for the proposed system.

The Analysis Group wanted a graphical interface between the "pilot" and computer built using a TEKTRONIX 4016 terminal. This terminal offers two modes of interaction: direct keyboard inputs and crosshair position/selection. Pilot inputs would be a combination of menu picks using the crosshair capability of the 4016 terminal and keyboard inputs to further define pilot actions when required. Output would be a graphical representation of a Radar Warning Display, a graphical representation of what the pilot could actually see through the aircraft windows, and a block containing cockpit instrument values and equipment status. In particular, the following objectives were established.

Pre-Simulation

In order to simulate an aircraft, a defense environment, and bombing mission parameters, a number of variables need to be initialized. The program would be most flexible if all the aircraft parameters were interactively alterable. This would in effect, allow any type of aircraft to be simulated. However, in the interest of simplicity we decided to make the flight characteristics static, that is, remain fixed at their preset values, but allow the ECM and weapon configuration to be interactively alterable before the simulation begins. Enhancement of the interactive adjustment to include more parameters will improve the implementation. In particular, the following structure will be used.

Preset Data

A) All aircraft performance capability (i.e. the "type" aircraft is constant)

B) The defensive environment (input from an Avionics Laboratory database)

C) An abort mission profile

D) A predefined flight path

Interactively Alterable Data

A) ECM configuration

   1) Number of Chaff pods

   2) Degrade factor for a chaff pod

3) Number of infra-red flares

4) Degrade factor for a flare

5) Number of radar jammers

B) Weapons Load

1) Number of "Iron" bombs

2) Number of "Smart" bombs

3) Number of RF missiles

4) Number of IR missiles

5) Probability of kill (PK) for each weapon type

<u>The Simulation</u>

<u>Interactive Inputs</u>

The aircraft would initially be a low level simulation only, to be enhanced as time permitted later in the effort. The menu pick would allow for pilot actions as follow:

A) Turns

Change direction, at normal or maximum rate

B) Speed

Increase/decrease, at normal or maximum rate

C) Altitude

Increase/decrease, at normal or maximum rate

D) ECM-selection

One/all

E) Bombing mode

F) Resume pre-set flight path

G) Mission abort

In addition, the menu picks could be further defined by keyboard inputs to determine the following:

A) Heading Change

B) Airspeed Change

C) Altitude Change

D) Which jammer

1) Direction

2) Frequency

3) Power

E) Which weapon

Which target

F) Simulation timestep - the time advance in the mission before the user is again allowed to make inputs

G) "Real time" factor - a means to accelerate or decelerate the real-time aspect of the simulation

Interactive Outputs

In order to simulate an aircraft with an active pilot, two requirements exist. The input data section above addresses the means by which the pilot controls the aircraft. The other requirement is to simulate the enemy threats upon which decisions and resultant actions are based.

The simulation feedback will be in four parts:

A) A computer generated Radar Warning Display which includes:

   1) a symbolic representation of the highest priority enemy threats (SAM sites, AAA, etc.)

   2) an indication of degree of threat based on rf signal strength and radio direction finding principles

   3) an indication of the type of threat being displayed (track, acquisition, or launch mode)

B) A computer generated visual representation of the threat environment to include:

   1) a symbolic representation of SAM sites/type within the pilot's visual capability

   2) a symbolic representation of AAA sites/type within the pilot's visual capability

C) Tabulated instrument readings which display actual vs. preplanned values for the following aircraft/mission parameters:

   1) elapsed time (hours:minutes:seconds)

   2) true heading (degrees magnetic)

   3) altitude (feet)

   4) ground speed (knots)

   5) fuel remaining (thousands of pounds)

6) total fuel flow rate (thousands of pounds per hour)

7) position (latitude, longitude)

D) Tabulated listing of the following aircraft equipment/status

1) jammer name/on-off, frequency selected, power, direction

2) chaff/number of pods remaining

3) bombs/type, number remaining

4) cruise missiles/type, number remaining

E) Interactive message area which will prompt the pilot for keyboard inputs and display his selections as appropriate (i.e. degrees of heading change for turn, knots of airspeed change, etc.)

Post Simulation

Since this project is an enhancement of the capabilities of already existing simulation software (the AADEM model), the previous output generating routines should be coupled to this project so that there is no loss of already existing reports. In addition, a graphical representation of the new flight path through the threat environment should be made available to the user on request. The generation of post simulation output will be considered a low priority requirement.

Required Functions

Numerous functions need to be performed in order to accomplish the tasks discussed above. In general, the capability to update all aircraft performance parameters, equipment status parameters, and defensive site status parameters on a continous time basis will be required. In addition, several classes of functions need to be performed. These include, but are not limited to the following:

A) To support the interactive pilot actions requires the following capabilities:

1) The capability to generate a menu of potential pilot actions

2) The capability to interpret user inputs in a unique manner,and to gracefully recover from illegal inputs.

3) The capability to prompt for further definition of pilot desires when needed.

4) The capability to modify the mission profile so that the "aircraft" correctly responds to "pilot" inputs

5) The capability to determine current aircraft position based on the combination of preset mission profile, pilot inputs and elapsed time

6) The capability to store, update and display aircraft status parameters

B) The interface between the existing Avionics Laboratory simulation and the man-in-loop simulation requires the following capabilities:

   1) The capability to interpolate aircraft position on a continuous time scale as desired from the discrete position vs. time values supplied by the predefined flight path

   2) The capability to determine relative position of the aircraft and air defense sites on a continuous time basis from the discrete data furnished in the defense environment data bases

   3) The capability to determine when either a defensive site or the aircraft has been destroyed

C) To generate the Radar Warning Receiver display and the pilot's visual display require the following capabilities:

   1) The capability to draw the various geometric figures which combine to build the display

   2) The capability to translate the X-Y viewport coordinates so that the aircraft and threats are displayed in the correct relative position

3) The capability to update the display periodically

The above outline describes precisely _what_ this system should do. The next step in software development is to determine _how_ the system will do it.

IV Software Design and Development

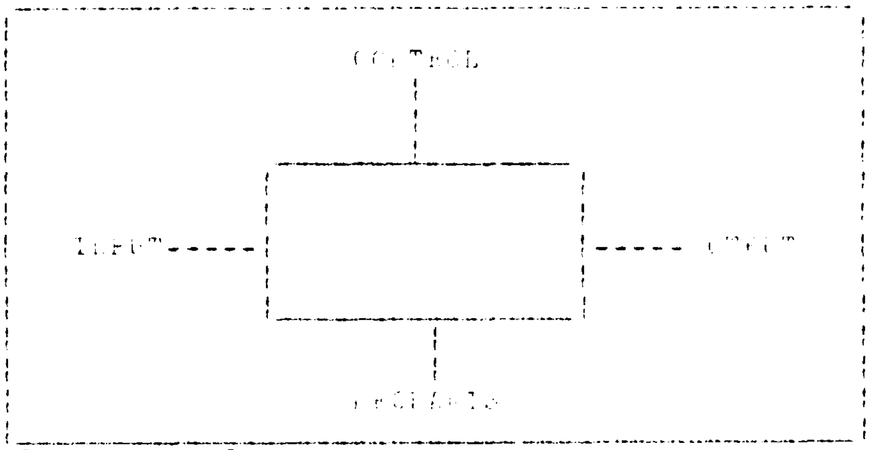
The growth in size and number of software system developments has disclosed a need for more rigorous means of designing and documenting software projects than has previously been used. Several "structured analysis" techniques have been developed as attempts to satisfy this need. Each of these techniques uses a set of charts or diagrams along with a brief verbal description to produce a more precise, more descriptive, and better structured definition than is possible with a plain English specification. The language used in each of these structured analysis techinques should make it possible to produce a complete, consistent, and unambiguous specification which is easy to understand and verify by both the people who originated the system concept and the designers responsible for developing a working system.

For this project I used a slightly relaxed version of the Structured Analysis and Design Technique (SADT) developed by SofTech, Inc. Some of SofTech's requirements are based on the situation where analysts design the system then programmers write the code. Since I was both analyst and programmer for this project, I feel that nothing was lost in my changes. The rest of this chapter describes briefly the mechanics of SADT as expressed in some introductory material from SofTech (Ref [illegible]). The final section of the chapter discusses this project specifically.

Description of SADT

A structured analysis model consists of a hierarchy of diagrams which describes the activities or functions of a system by breaking down the high-level functions into a series of progressively more detailed subfunctions. Each diagram represents a single, self-contained activity which is part of an overall, higher level function. A diagram shows how an activity is decomposed into sub-activities and how these sub-activities relate to each other. By decomposing each sub-activity on a high level ("parent") diagram into a group of next lower level ("children") diagrams, the designer produces a collection of charts which make up a tree structure description of the system.

Each diagram is made up of its title, node number, boxes, arrows, and descriptive words. The title should concisely express the diagram's overall function. Each box (referred to as a node) on an activity diagram represents a sub-activity of that diagram's function. Arrows represent data; in this case data means anything that is not an activity. The side of the box to which an arrow connects is critical, and expresses what type of data is being represented. The various meanings for an arrow are given in figure 2.



[Ref 10:1-3]

Figure 2
Box/Interface Arrow Definition

An input is data which is converted by the activity into output. A control is data which may or may not be changed by an activity, but acts as a constraint on how an activity converts inputs into outputs. A mechanism is a processor which performs the activity of the box.

Arrows show the interrelationship between boxes on one or more diagrams. As such, they may either merge into a more general category or split into more specific categories. Unless labeled with a more specific data definition, both branches of a split arrow are assumed to represent the same data. It is also possible for an arrow to split or merge to indicate an exclusive "OR" condition. That is, the data reflected on the single arrow at an "OR" point is one, and only one, of the pieces of data represented by the multiple arrows at the point. Figure 3 summarizes the SADT arrow conventions.

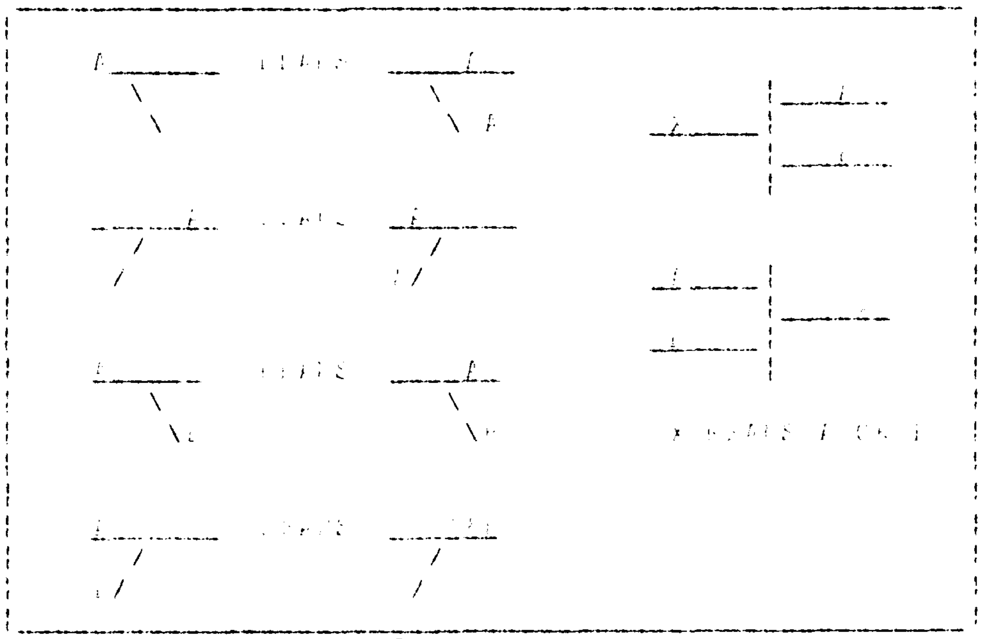

[Ref 13:3-13, 3-15]

FIGURE 3
SADT Arrow Conventions

A numbering scheme is used to distinguish between nodes/activities on a single level and those on different levels in the activity model. Each diagram is given a node number (e.g. A-0, A5, A23, etc.) based on its position in the tree structure of the model. By convention, the top level context diagram has node number A-0 (A minus zero) and its child diagram which is the top level decomposition of the system has node number A0. Each box on the A0 diagram and all lower level decompositions are numbered. Each lower level diagram is given a number made up of the parent diagram's node number followed by the number of the box being decomposed. For example, the decomposition of box 2 from diagram A3 would have node number A32.

A node called ICOM (Input, Control, Output, mechanism) is used to connect those arrows which appear on both the parent and child diagram. The code is made up of a letter (I, C, O, or M) followed by a number. The prefix explains which side of the box the arrow attaches on the parent diagram (see Fig 2) and the numbering scheme is left to right, top to bottom. The ICOM codes are implicit on the parent diagram from position, but are explicitly written on the corresponding one-ended arrows of the child diagrams. Note that an arrow, say I1, may appear on several diagrams and it have a different meaning in each case. The labels tie a child to its parent, but do not transfer beyond that context.

The only remaining factor about SADT is how to know when you're done. There are three general guidlines as to when the decomposition of a subsystem has been carried far enough. First, when it is impossible to tell anything more about what is to be done without deciding how to do it, stop. Second, when a box is reached which can be satisfied by some known tool, stop. Third, when a box is reached that has (or is expected to have) very similar counterparts at other places in the model, stop for now; possibly later a single general module can be built to handle all similar situations. Using SADT principles and techniques as discussed above, and based on the system specification outlined in Chapter 3, I developed the following system design description.

System Design

The high level description discussed in Chapter III breaks neatly into three modular pieces: processing input and output, simulating the mission, and printing statistics as to mission successes and failures. From the above observation, Node A0 for the proposed system was drawn (Figure 4).

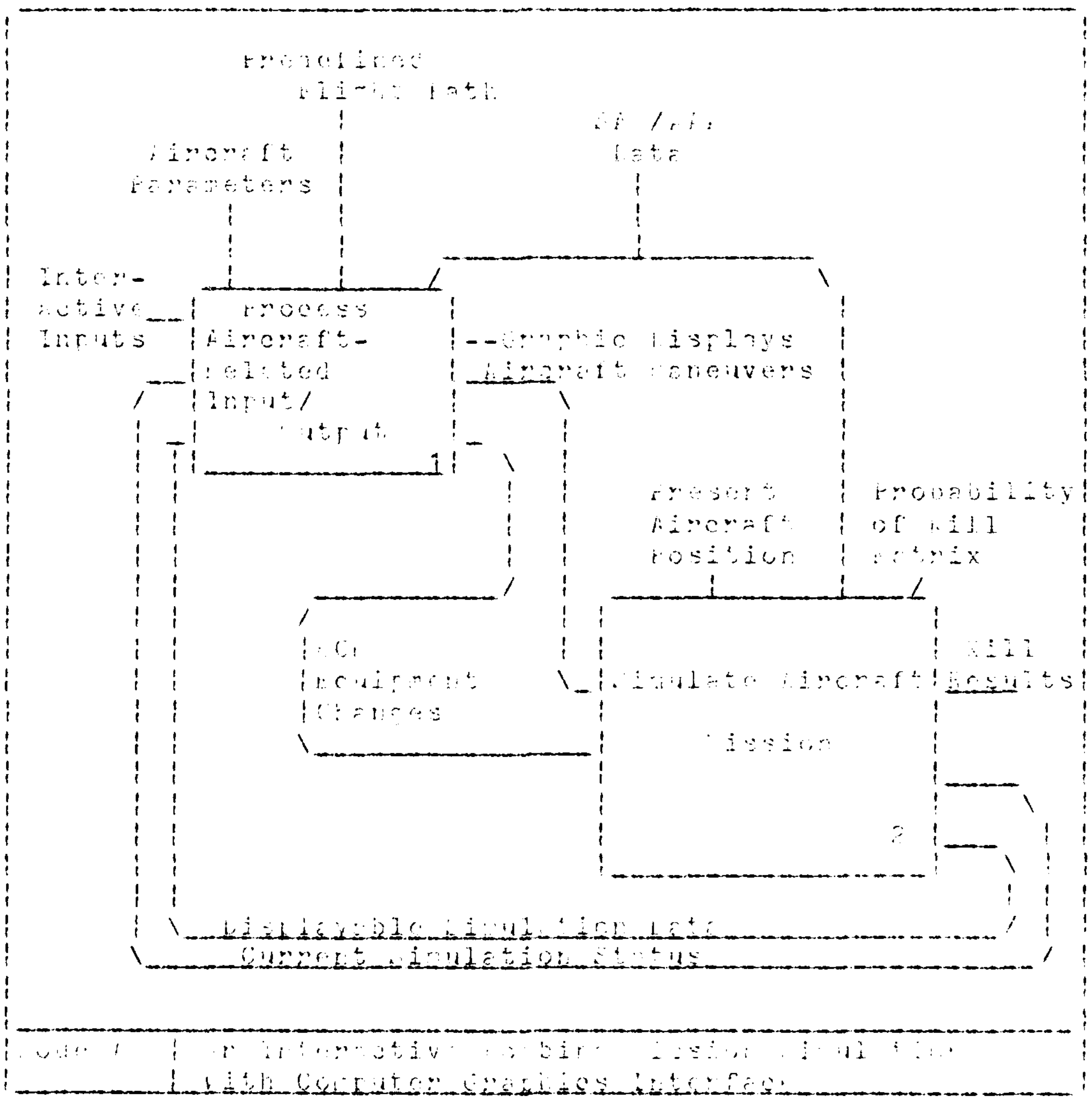

Predefined Flight Path
Aircraft Parameters
Data
Inter-active Inputs
Process Aircraft-Related Input/Output
1
Graphic Displays
Aircraft Maneuvers
Present Aircraft Position
Probability of Kill Matrix
ECM Equipment Changes
Simulate Aircraft Mission
2
Kill Results
Displayable Simulation Data
Current Simulation Status
Node A0
An Interactive Combined Mission Simulation with Computer Graphics Interface

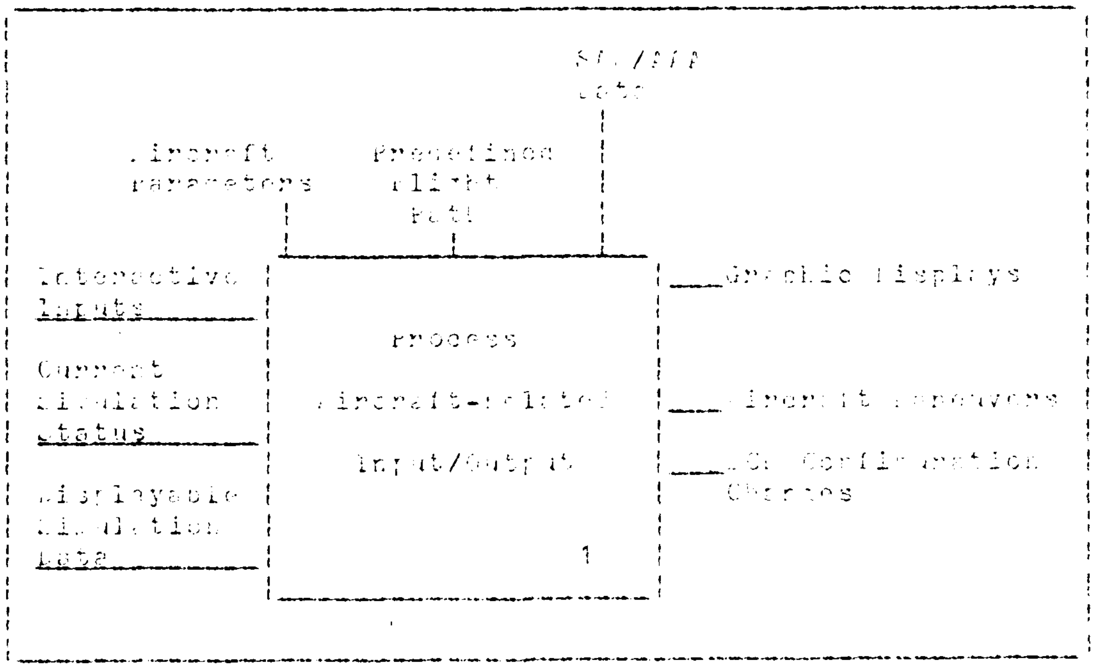
Figure 4
Node A0

The output from box 2 level will result [illegible] post-simulation output routines, but these are not shown since the project does not involve some of post-simulation output explicitly.

Figure 4, represents a complete high level description of the proposed thesis project. As stated above, this does not include post simulation summary output. For example, from an aircraft's perspective (remember we are dealing with a bombing mission) the constraints are: the threat environment, the aircraft's present position, the aircraft's capabilities, the directed route of flight, and the performance probabilities of the aircraft and enemy weapon systems. These are shown as control arrows on Node A0. As an example of processing, consider an interactive input, say descend 500 feet. The aircraft parameters of descent rate and fuel use rate are controls used to convert this input into a legitimate aircraft maneuver. This maneuver now becomes an input to box 2. The maneuver, constrained by the present aircraft position, takes up the current simulation status, which is passed back to box 1 and also displayed to the pilot. In like fashion, all arrows can be established, and the need for all arrows can be verified. It is important to check two requirements: that everything on the routine is needed, and that everything needed is on the routine.

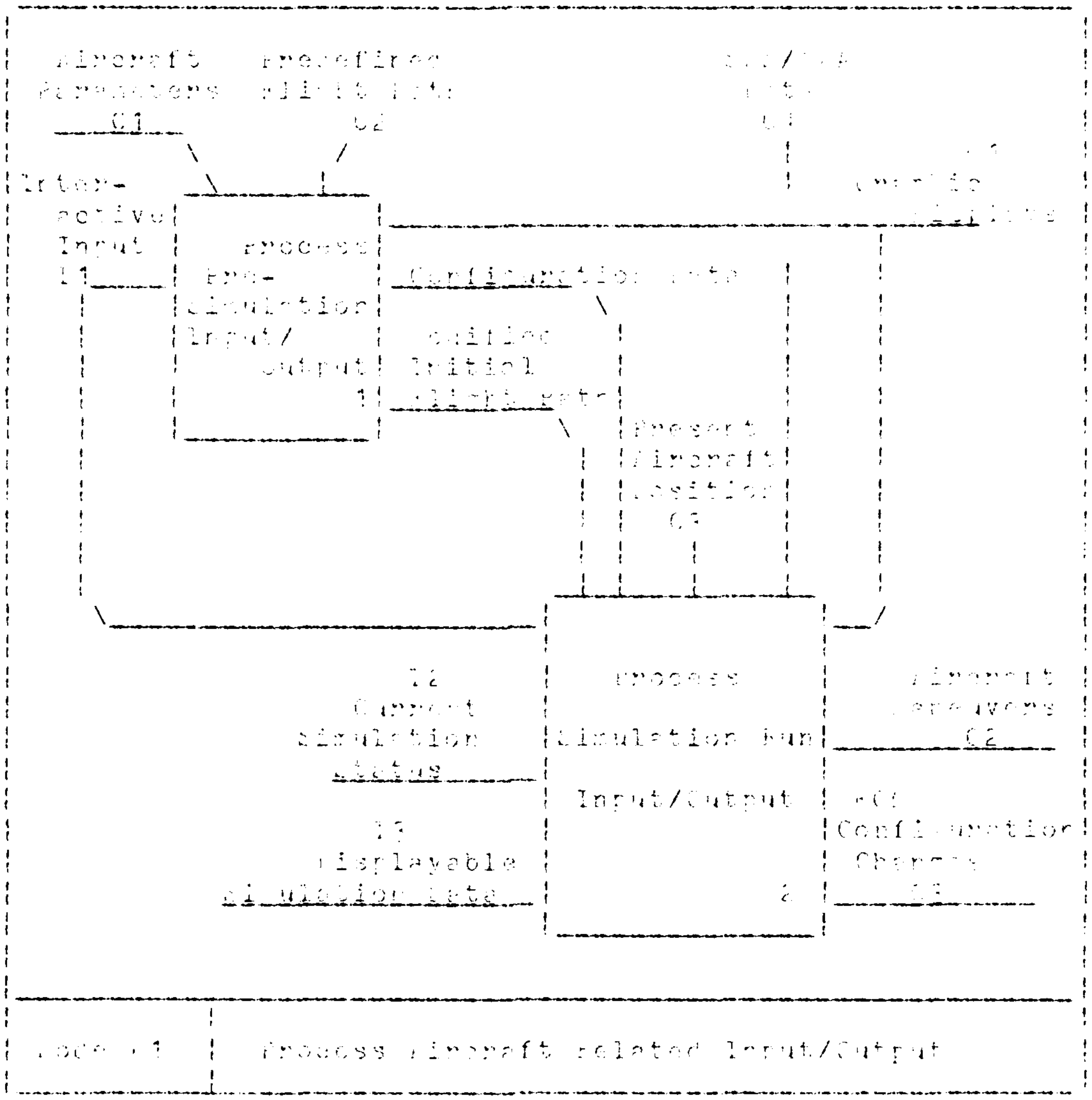
After testing for consistency and logical [illegible] level, we check to see if we are done. [illegible] expect to decompose further from here, but just for demonstration recall: There are two indications of when we need not decompose further. We can tell more about what the system will do, before we get into how, so we fail the first test. Secondly, if there was already a known tool to satisfy this description, I wouldn't need to build one, so we also fail the next test. We have not decomposed the system at all yet, so the third test does not apply. Clearly there is more to do.

Figure 5 would not be included in a normal SADT description, but is shown here to smooth the transition from A0 (Figure 4) to A1 (Figure 6) for the reader.



SIM/ATA Data
Aircraft Parameters
Predefined Flight Path
Interactive Inputs
Current Simulation Status
Displayable Simulation Data
Process Aircraft-related Input/Output
1
Graphic Displays
Aircraft Maneuvers
A/C Configuration Changes

Figure 5
Process Aircraft-related Input/Output



Aircraft Parameters C1
Predefined Flight Data C2
Interactive Input I1
Process Pre-simulation Input/Output 1
Configuration Data
Modified Initial Flight Data
Present Aircraft Position O3
Current Simulation Status I2
Displayable Simulation Data I3
Process Simulation Run Input/Output 2
Aircraft Maneuvers O2
Configuration Changes
Node A1
Process Aircraft related Input/Output

FIGURE 6
Node A1

remember the numbering scheme is top to bottom, left to right; the ICOM code starts at [illegible]. Thus from Figure 6, Interactive Inputs will be I1 on the A1 diagram, predefined flight data will be [illegible] Aircraft Maneuvers will be O2. In a fashion similar to [illegible], I [illegible] level, [illegible] structure was complete, applied the above tests, and decomposed further when appropriate. The complete [illegible] description of the project is included as Appendix II for the interested reader.

From the [illegible] description, for a large project, it is sometimes useful to do flowcharting on a level decomposition basis also. Since most readers are very familiar with flow charts, I will merely offer Figure 7 and decomposition figures [illegible] high level view of the control structure of the program. From this high level structure, the modules can be visualized, and lower level flowcharting and coding can begin.

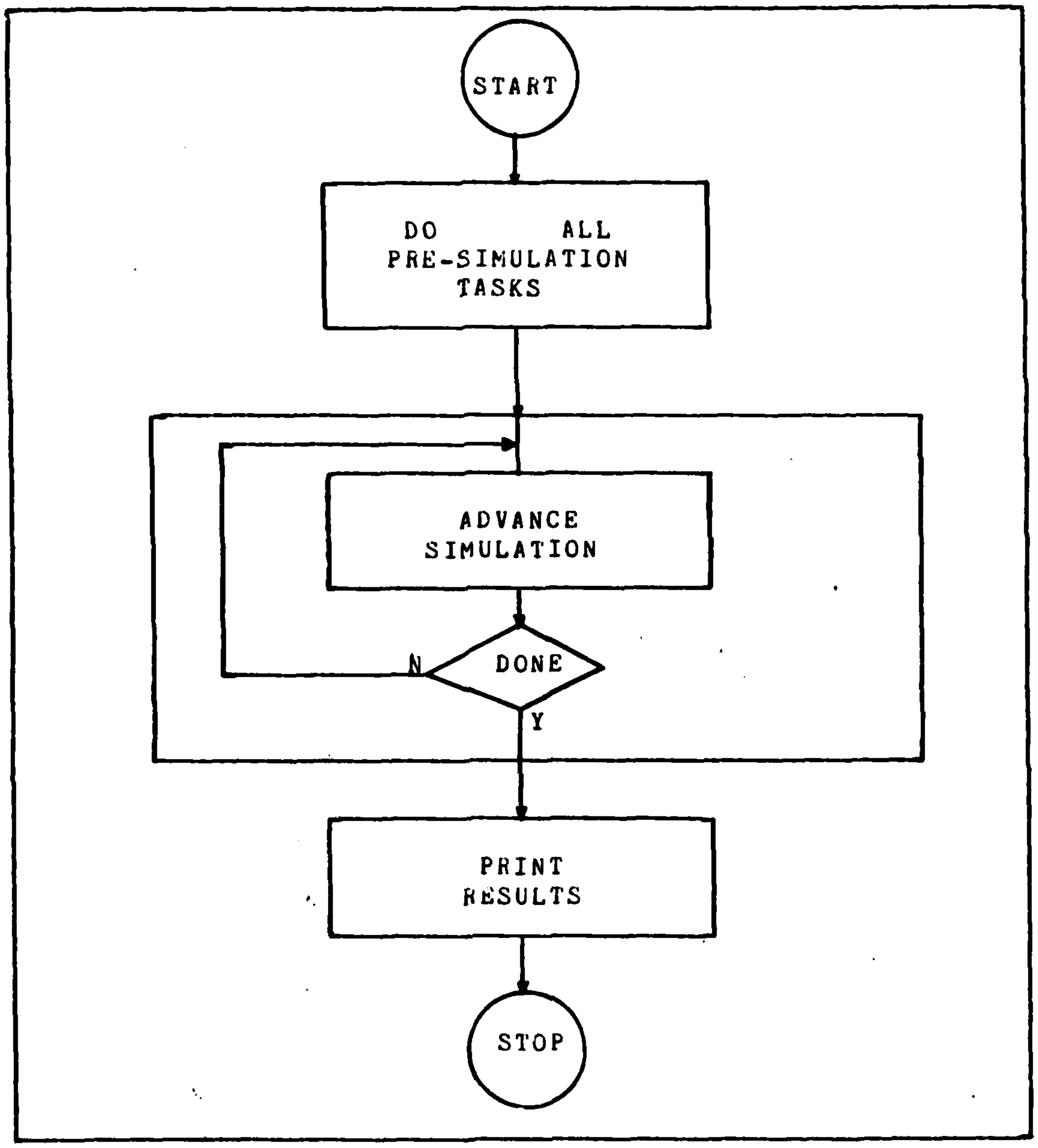


FIGURE 7
Top Level Flowchart

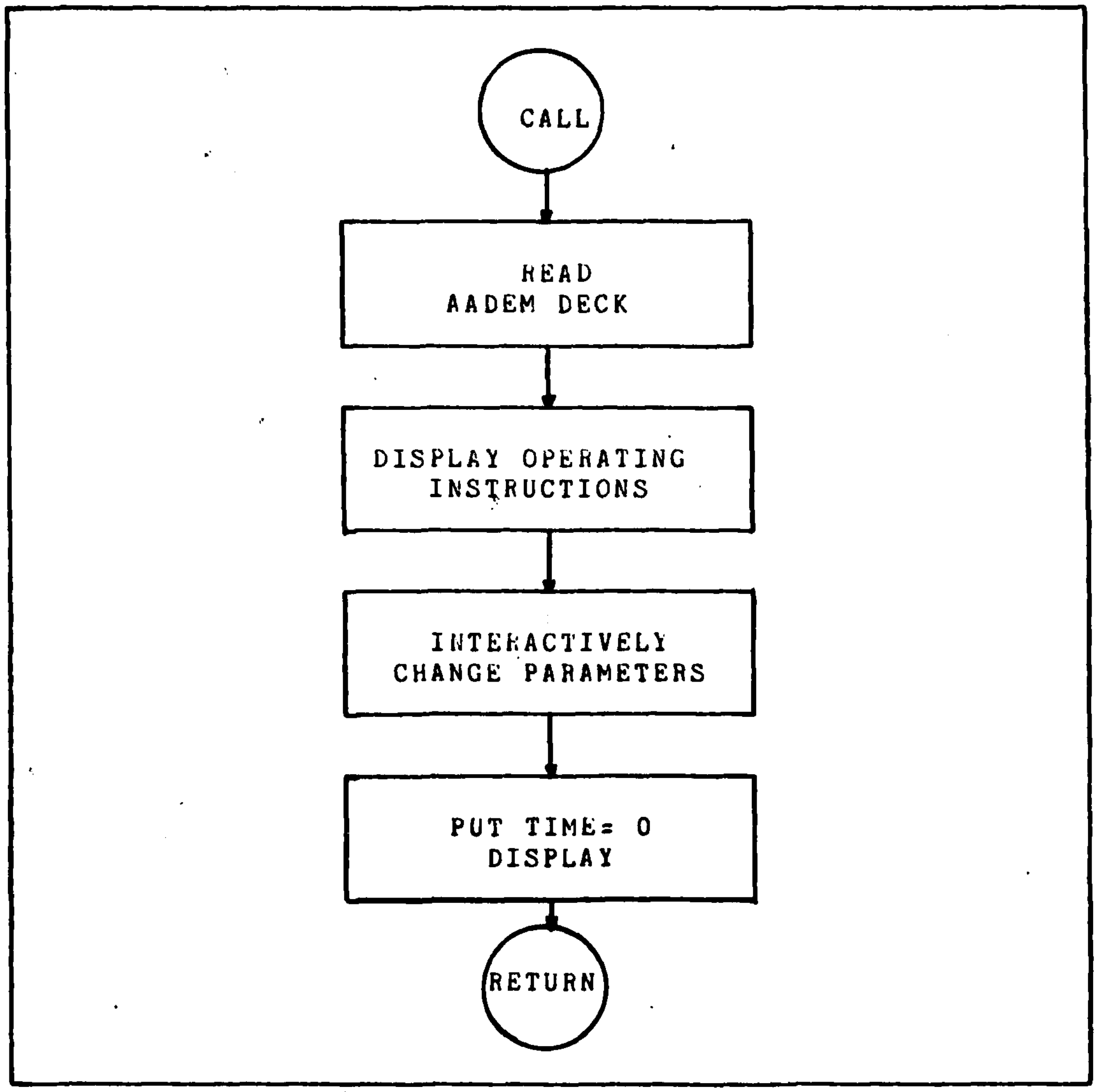

CALL
READ AADEM DECK
DISPLAY OPERATING INSTRUCTIONS
INTERACTIVELY CHANGE PARAMETERS
PUT TIME= 0 DISPLAY
RETURN

FIGURE 8
Do All Pre-simulation Tasks

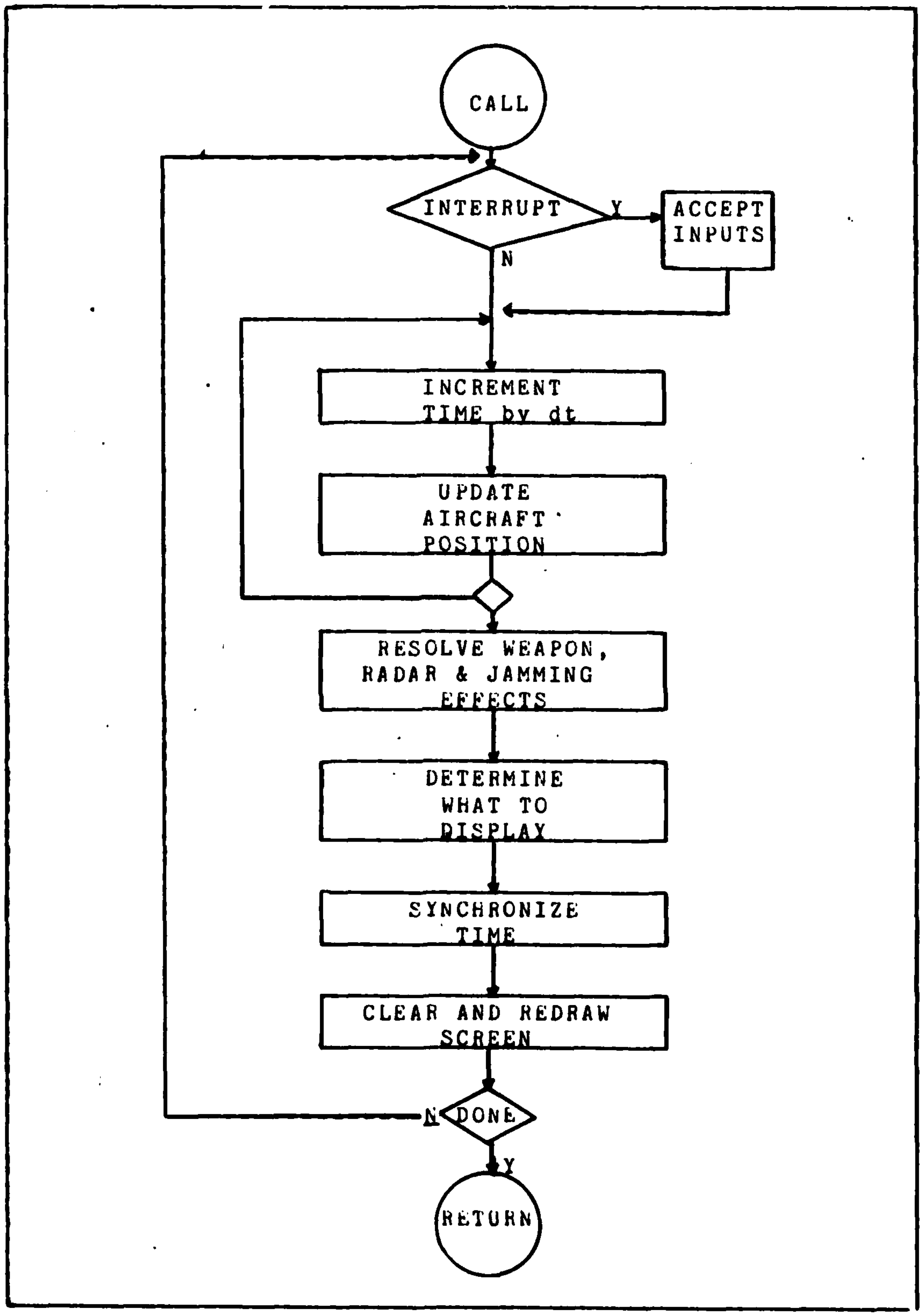


FIGURE 9
Execute Simulation

Algorithm Design

Most algorithms used in the simulation were structured after the AFIT model, however a few adjustments were needed to support the "real time" aspects. For example, the AFIT model updates the aircraft position, resolves weapon effects, radar vs. jammer effects, and the other simulated activities on a twenty-second simulated timestep, but makes no effort to make that timestep approximate twenty clock seconds. This is because in the AFIT model all parameters and scenarios are established at compile time, and no "real time" observation of the activities is implemented. For the interactive model, it was desired to observe the activities on a user determined timestep, and to make the timestep approximate clock time. It was also desired to use the basic position formula:

New parameter = Old parameter +

(rate of change per time unit * time change)

or: $X' = X + (dX/dt) * dt$

and ignore second order terms. In particular the following equations are used in subroutine STATAL:

a) If the aircraft is climbing

ALT= ALT + [illegible] * DT

or ALT= ALT + [illegible] * DT

b) If the aircraft is descending

ALT= ALT - [illegible] * DT

or ALT= ALT - [illegible] * DT

a) If the aircraft is accelerating

SPEED= SPEED + SPDACC * DT

or ALT= ALT + ALTACC * DT

b) If the aircraft is decelerating

SPEED= SPEED - [illegible] * DT

or [illegible]= [illegible] - [illegible] * DT

c) If the aircraft is turning

HDG= HDG ± [illegible] * DT

or HDG= HDG ± [illegible] * DT

d) Aircraft position

ACX= ACX + SPEED * SIN(HDG) * DT

ACY= ACY + SPEED * COS(HDG) * DT

e) Fuel remaining

FUELTOT= FUELTOT - FUELRT * DT

An error analysis for this approach, which is included in Appendix 1, shows that the error should be less than four percent, for a dt of one second at typical aircraft rates of change. Note that when the aircraft is in stable flight (most of the time) there is no error. By decreasing dt, the associated error would be less.

Advancing the simulation is a two step process. First, the simulation [illegible] speed, altitude, heading and the X and Y coordinate values based on the present value, rate of [illegible] [illegible] [illegible]. This [illegible] is [illegible] until [illegible] [illegible] is exhausted. [illegible] [illegible] [illegible], [illegible], [illegible] [illegible], [illegible] [illegible] [illegible] [illegible].

the [illegible] (that is, once per observation timestep). When the above is complete, a "wait" function is used to synchronize the observation timestep of elapsed time with clock time. [illegible] time has been re-synchronized, the display is cleared, then redrawn to reflect the new [illegible].

The above "[illegible] simulation" algorithm contains a test for user interrupt. So long as the user has not indicated [illegible] desire to make an input, the above algorithm is repeated each observation timestep. When the user interrupts, the screen is drawn to reflect current simulation status, a time out mode is entered, and the user is prompted for inputs. The simulation timing is stopped while the user makes inputs, then restarted when the user signals that inputting is complete. When all inputs have been serviced, time is restarted, and the above timestep loop is re-entered. With the desired system now fully designed, and the algorithms determined, the program can be coded.

The [illegible] Code

As stated previously, computer models tend never to be finished. Further enhancement and development of some aspect [illegible] will [illegible] representation of the "[illegible] model." [illegible] will continually be modified, I [illegible] project into fairly small, functional modules. [illegible] well defined function - no more, no less. This style of coding is recommended by [illegible] [Ref 10], Yourdon and Constantine [Ref 11], and several others. There are numerous advantages to this style of writing. First is ease of enhancement and change, since to enhance a function requires rewriting only the one module which performs that function. Second is ease of debugging, since if a given task is not being performed correctly, the module which should perform that task is easy to locate. Finally, testing is considerably more straight forward since the problem is broken into many well defined parts, with limited and precise outcomes. The disadvantages to this style of coding are twofold: It always takes more memory space in the computer because of the extra control overhead, and it often takes slightly longer to execute.

The program will ultimately be run with classified data, which requires that the computer be dedicated to this one task (i.e. not multiprocessing) while it is in operation, so memory space is not a constraint. For now, extra timing delays are being used to slow execution to "real time." If [illegible] pushes execution [illegible] significantly past "real time," modification [illegible] be required later.

Writing many function modules also tends to turn the program into a very "bushy", deep tree. In fact, the program is [illegible] into more than seventy-five subroutines, not including the interface with only [illegible] the AIDE model, the project is well over one hundred subroutines. Figure 10 gives the first few levels of the tree. The interested reader can reference these in the data dictionary, Appendix III, and build the [illegible] up as desired.

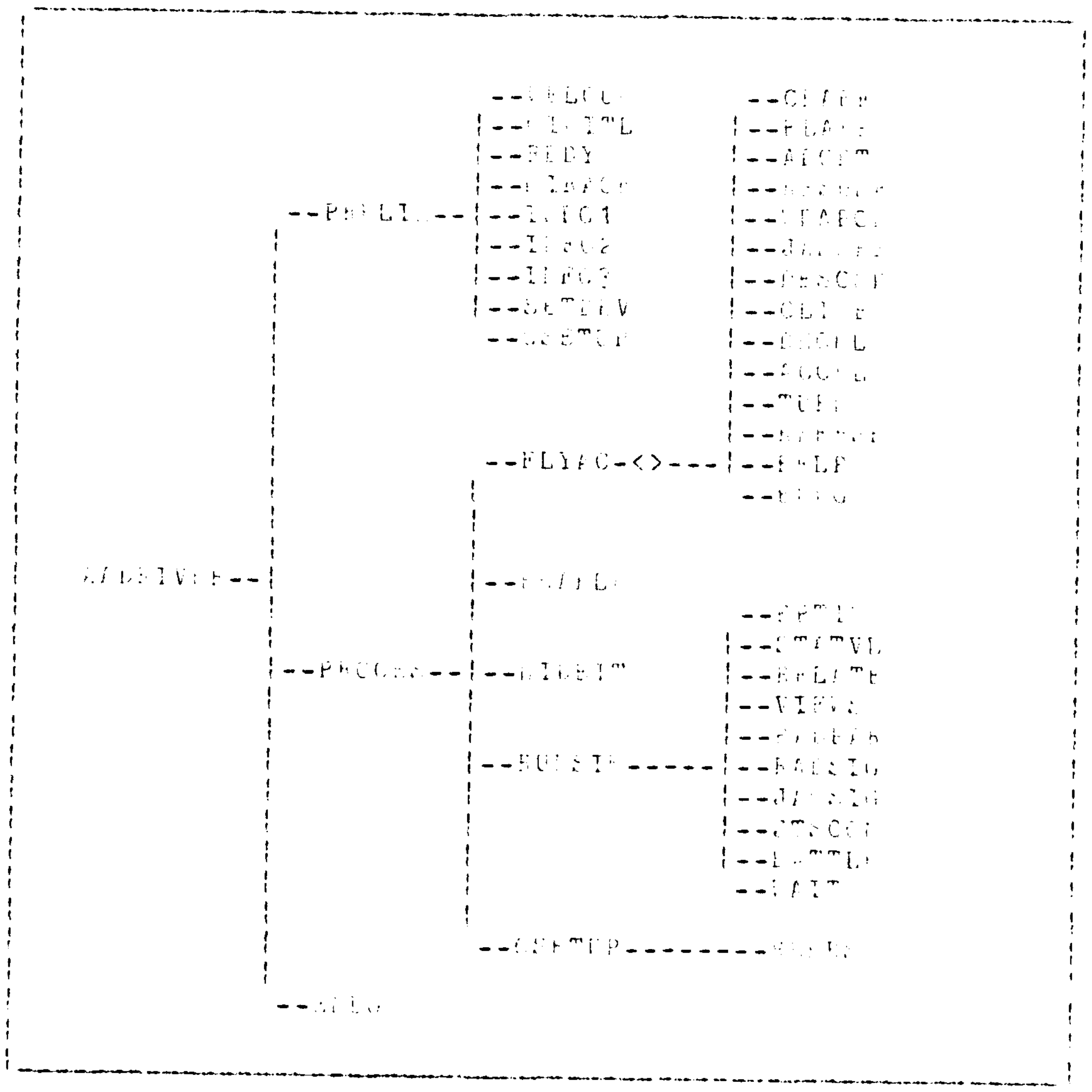


Figure 10
"Top" of the Subroutine Tree Structure

Data Structures

The data structures used in this program are extremely simple. With only a few exceptions, every data item is given an independent variable name. The exceptions are in three classes. The first class is for all words and phrases used in the display. These are printed from one-dimensional arrays of ASCII equivalent integers using the ANSTR procedure of PLOT-10. The second class is for jammer information such as power, sector, and frequency subscripted by the jammer number (from one to five). The final class is one two-dimensional array [illegible] (6,2). This array stores flags for flight maneuvers in progress (climb, turn, etc.), used in highlighting the menu. This will be discussed in more detail later.

Because the aircraft simulation was limited to two rates of change (normal and maximum) in each direction, (horizontal, vertical, lateral) these variables are not subscripted. Enhancing these rates of change with more values will lead to subscripting and a table look up scheme later. The common blocks are functionally divided in a manner similar to the subroutines, and their names express what variables they include. The FORTRAN restriction on number of characters in variable names sometimes makes them appear cryptic, but references to the Data Dictionary (Appendix III) will tell you what words have been abbreviated to form the name.

For example, [illegible] (aircraft

parameters), includes [illegible], [illegible], [illegible], [illegible], (and others) which are the normal descent, maximum descent, normal deceleration, and maximum turn rates respectively. The common blocks are all located in subroutine CINITL for ready reference. (The C differentiates my routine from several initialization routines used by the [illegible] model.) Also, any name that is confusing can be cross referenced through Appendix III.

Implementation

When this simulation design was implemented, it was code named MIRAGE for short. The acronym stands for Man-in-loop Interactive Real-time Aircraft mission simulation using Graphics to display the Environment. Execution of the MIRAGE system is divided into two parts: pre-simulation and simulation. During the pre-simulation phase, all input files are read, common variables are initialized, a summary of the user's guide (Appendix IV) is displayed at the user's option, and the aircraft is interactively configured to the user's specification. These functions are accomplished using FORTRAN WRITE commands for displays and prompts, and FORTRAN READ commands for user responses. This section of the program is typical of interactive menu-driven dialogue software.

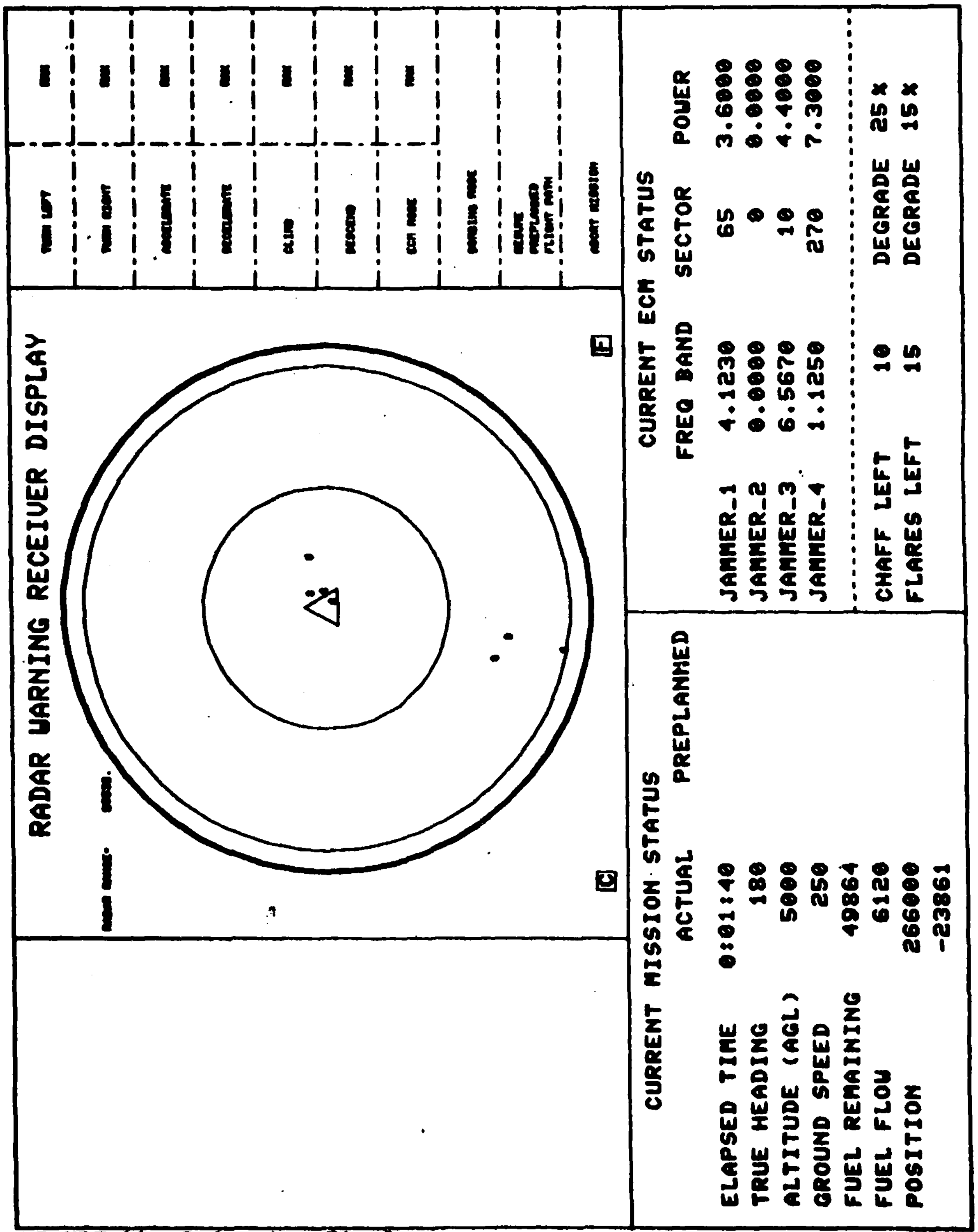
The second phase is the actual simulation. It is meant to operate in real time when possible, but is in [illegible] state during [illegible] interaction. This is accomplished using an interrupt handler, timestep, wait function, and delay multiplier. The timestep is to accommodate the problem of interface using a storage tube display such as the TEKTRONIX 4014. The timestep controls how far the simulation advances between the clearing and redrawing of the screen. If activated, and not interrupted, the simulator will update all parameters, wait for clock time to approximate simulated time, clear and draw the appropriate display with parameters now current, and repeat. Recall that overall the simulation is a two stage process (see Figure 9). The first stage utilizes a loop which updates the aircraft position in repeated small steps, to obtain its present position. In the second stage, this aircraft position is then used in resolving weapon, radar, and jamming effects. The displayable data is then determined, and after waiting to synchronize the timing, the displays are drawn. If interrupted, simulated time stops, and the user may enter maneuvers.

Entering aircraft maneuvers is a combination of picking from a menu with the TEKTRONIX cursor, and further defining the maneuver by responding to prompts. In addition, there are control characters available to execute functions outside the realm of "flying the combat mission." These include refreshing the screen, [illegible] the timestep or delay multiplier, displaying a help message, stopping and restarting the simulation, or terminating execution of the program.

The delay multiplier is used to alter the real time aspect of the simulation. The delay is the clock time between how long it takes to update all simulation parameters, and the timestep. This is the input parameter to the wait function. You can accelerate the simulation by multiplying the delay by a number less than one. You can slow the simulation by using a number greater than one.

There are two displays available depending on the strategy of the user at the time. The defensive display (Figure 11) gives a Radar Warning Receiver and ECM Status. The offensive display (Figure 12) gives a top down visual presentation and weapon status. The other three parts, the interactive scratch pad at upper left, interactive menu at upper right, and mission status at lower left are always presented.

Given this general description of the program structure, I will go on to discuss the testing and verification. For more information on exactly what the program can do, or how it does it, I refer you to the User's Guide, Appendix IV.



CURRENT MISSION STATUS

	ACTUAL	PREPLANNED
ELAPSED TIME	0:01:40	
TRUE HEADING	180	
ALTITUDE (AGL)	5000	
GROUND SPEED	250	
FUEL REMAINING	49864	
FUEL FLOW	6120	
POSITION	266000 -23861	

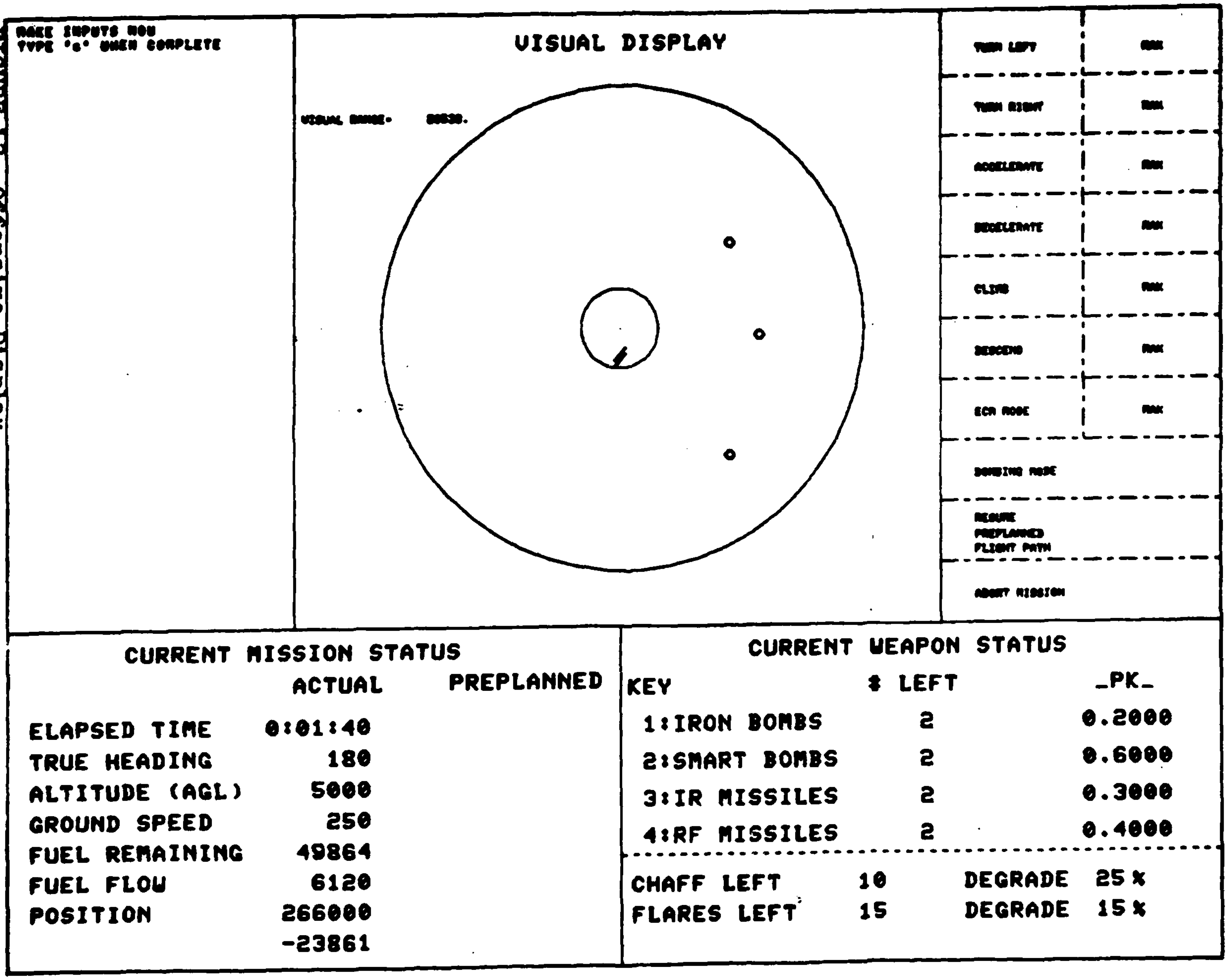
CURRENT ECM STATUS

	FREQ BAND	SECTOR	POWER
JAMMER_1	4.1230	65	3.6000
JAMMER_2	0.0000	0	0.0000
JAMMER_3	6.5670	10	4.4000
JAMMER_4	1.1250	270	7.3000

CHAFF LEFT	10	DEGRADE	25%
FLARES LEFT	15	DEGRADE	15%

FIGURE 11 Defensive Display

9


MAKE INPUTS NOW
TYPE 'C' WHEN COMPLETE

VISUAL DISPLAY

VISUAL RANGE= [illegible].

TURN LEFT	MAX
TURN RIGHT	MAX
ACCELERATE	MAX
DECELERATE	MAX
CLIMB	MAX
DESCEND	MAX
ECM MODE	MAX
BOMBING MODE	
RESUME PREPLANNED FLIGHT PATH	
ABORT MISSION	

CURRENT MISSION STATUS

	ACTUAL	PREPLANNED
ELAPSED TIME	0:01:40	
TRUE HEADING	180	
ALTITUDE (AGL)	5000	
GROUND SPEED	250	
FUEL REMAINING	49864	
FUEL FLOW	6120	
POSITION	266000	
	-23861	

CURRENT WEAPON STATUS

KEY	$ LEFT	_PK_
1:IRON BOMBS	2	0.2000
2:SMART BOMBS	2	0.6000
3:IR MISSILES	2	0.3000
4:RF MISSILES	2	0.4000

CHAFF LEFT	10	DEGRADE	25 %
FLARES LEFT	15	DEGRADE	15 %

FIGURE 12 Offensive Display

V Testing and Verification

Testing is the process of executing a program with the intent of finding errors. This implies that a good test case is one that has a high probability of detecting an as yet undiscovered error. The first section of this chapter will discuss the principles of testing in general terms. The second section will then discuss the thesis project in particular.

In General

Testing can be divided into two main classes: "black box" testing and "glass box" testing (sometimes referred to as "white box" testing). In black box testing, the test data is derived solely from the project specifications. Often the test cases for a software project are written before or during the actual software development. These tests are mostly to insure that the software does everything it is specified to do.

For example, suppose a module is supposed to sort up to fifty names, each with up to twenty characters, into alphabetical order. The most common first test would be to feed it a list of names and be sure that the output is alphabetized. Another common test would be to ensure that it handles too large an input file (say 51 names) reasonably. Also, does it handle the cases of zero or one name? These are the boundary conditions, and should always be tested. Often in testing, due to the potentially large number of possible cases, situations are broken into equivalence classes. That is, if the sort routine gracefully handles 51 names, hopefully the same error handler is called for larger files, so 51, 52, ... are equivalent in a sense, and they probably don't need to be tested individually. Similarly, if the routine correctly sorts 2, 25, and 50 elements, chances are it will handle the numbers between. The number of characters in the names would be tested in the same way as the number of names in the file.

This brings up a point on efficient testing. Whenever a test is supposed to work according to the specifications, any number of situations may be tested simultaneously. So we could test a name with only one character, one with twenty, and a list of fifty names all in the same run, and expect an alphabetized list. However, when our test is designed to be outside specifications, each case must be handled separately. This is to ensure that each invalid case is correctly handled. And finally, the tester must always know the expected output before executing the test. It is very easy to look at output that is close to correct, and assume it is perfect.

The other class of testing is glass box testing. These tests take advantage of knowledge of the internal structure of the code and the program logic. They look at subscript ranges, divisors, loop control, etc., and are designed to find overlooked situations which will cause problems.

Both classes of testing often incorporate "error guessing." An experienced programmer considers the problem at hand, tries to guess the potentially troublesome cases, then tests that the program adequately performs these cases.

Program testing is where the programmer is paid back for segmenting code into routines which perform one well defined function. Each subroutine can be tested using the techniques discussed above by calling it with a simple driver. If the function is performed correctly, the only question remaining is whether the tests were adequate. If the function is not performed correctly, there is no question as to which routine is at fault. When a given routine passes its tests, other routines at that level are similarly tested. The next step is to replace the driver with the actual routine that calls the given level, and test in the same fashion with a driver at the next higher level. Continue to work in this fashion from the bottom up. This method strives to ensure that the "worker" modules are correct before the interactions of other modules confuse the issue. If each level is thoroughly tested, when the top is reached, the system is thoroughly tested. Unfortunately, with only a few specific exceptions, no one has developed a scheme to guarantee that a given set of tests is sufficient, while being less than exhaustive.

Before moving to the specifics of my project, there are a few other principles of testing often highlighted in the literature: [Ref 4, 8, 11, 12, 15]

A) A necessary part of a test case is the definition of expected results.

B) A programmer should avoid attempting to test his or her own programs.

C) The probability of more errors in a section of a program is proportional to the number of errors already found in that section.

D) Examining a program to see if it does not do what it is supposed to is only half of the task - the other half is seeing whether the program does what it is not supposed to do.

With the above philosophies, techniques and tools, we're ready to test the simulation.

The Simulation

The basic rule of software testing - avoid testing your own work - was unfortunately impossible to follow for this effort. I did, however, attempt to test the project thoroughly. There are eleven general functions which make up this software:

1) Text displays are presented on the screen

2) Parameters are input from files

3) Parameters are i[illegible]

4) Lines which are the same for every display are drawn

5) Words which are the same for every display are written

6) Lines for the offensive <u>or</u> defensive display are drawn

7) Words for the offensive <u>or</u> defensive display are written

8) Interactive commands are decoded

9) Parameters are updated

10) Parameters are printed

11) Site information is graphically displayed

It would be tedious and very dry to explicitly state every test case, reason for use, and expected outcome for each subroutine, so I will describe the testing in slightly higher terms for the higher level functions described above.

1) <u>Text Displays</u>

All text displaying routines are merely a call to set the character size, then a sequence of FORTRAN print statements. No parameters are input, and no variable are changed. For these modules, exhaustive testing is simply to read the output for typograhical errors, and format on the screen.

2&3) Input From Files

All routines which read input files to initialize variables were tested by using a parallel routine with shared common areas and write instead of read statements. When all variables written were the same as the variables read, I considered the input routines to be sufficiently tested.

4&5) Static Display Drawing

The static line drawing and word writing routines are in the same category as 1 above. When the lines are in the desired places with the correct words, in the proper position, in the proper text size, the testing is exhaustive.

6&7) Dynamic Display Drawing

Testing the static drawing for the offensive display (Visual Display and Current Weapon Status) or defensive display (Radar Warning Receiver and Current ECM Status) is only a small step from 4 & 5 above. Each subroutine was first tested with a driver until the words and lines were as desired. When this was satisfactory, they were coupled to the refresh driver, to check that the two displays toggled back and forth correctly.

8) Interactive Command Decoding

The interactive command handling routine was coded and tested in very small steps. Testing was no more complicated than for the situations above, but since there are seventeen boxes in the menu, two buttons, and six control characters, it took considerably longer. My routine calls the cursor, checks for a control character, checks for the space bar, and if none of the above, loops back to call the cursor. I tested every character on the keyboard to insure that none of the ASCII codes would be misinterpretted. Since the cursor handler of PLOT-10 returns a single character, plus the X, Y coordinate of the point, I believe that it is unnecessary to worry about combinations of characters causing problems.

For each control character, and the space bar, I initially had the routine write a message when selected. When I was convinced that this structure was correct, I did the same for the X, Y picking of buttons and menu boxes. When the exclusive-or structure was confirmed I replaced each message write statement with a call to a subroutine with the respective write message. When each box and button was tested this way, I went on to write a functional module to perform the respective task. These functional modules will be discussed next.

9&10) <u>Updating and Printing Parameters</u>

There are two types of parameters to update during the simulation: "counters" and "positions." The counters include chaff, flares, and weapons remaining, elapsed time, and fuel flow. Computing these involves simple subtraction for items remaining, addition for elapsed time, and periodic addition and/or subtraction to fuel flow despending on what maneuvers are taking place. Testing involves boundary testing to preclude negative numbers of items remaining, and correctness testing for the combinations of events. "Position" parameters include X position, Y position, fuel remaining, ground speed, altitude, and heading. As mentioned previously, all of these values are computed by the scheme:

New value = Old value + (change rate * time).

Insuring the needed common blocks were available, and that the specific formulas had no typographical errors was the crucial part of this testing. The simulation was repeatedly advanced by one second and stopped so that each parameter coud be checked through several cycles and a representative cross section of maneuvers. Since the printing of these parameters is done by converting the internally stored values from integers and real numbers to character strings, then using graphics text output features from PLOT-10, any incorrect output could have been caused either by the conversion routines or the arithmetic computation. I tested the boundaries, and error guessed when appropriate, trying to cause the routines to fail. I believe these routines are correct.

11) Displaying Sites

Displaying the sites is done for the Radar Warning Display and the Visual Display. In the Visual Display, it is a matter of accessing the X, Y coordinates of the sites from the AADEM model. I first wrote a test routine which accessed the data and wrote the site coordinates and the respective type array (SAM, AAA, etc.). I manually plotted this information on graph paper, and wrote a driver which read aircraft position and heading, then drew the associated visual display. I manually overlayed a visual range circle on my graph paper and rotated it to correspond to the aircraft position and heading. I compared my hand drawing to the computer display for a representative cross section of positions. A similar scheme was used to test the RWR routines, except the manual work was more complicated. It required computing the received signal strength for numerous sites, translating this into the X, Y positions to plot on the RWR, plotting the computations and comparing the display to the graph paper. I tested for all sites displayed, none displayed, and intermediate situations combined with various headings and altitudes. I believe these were sufficiently tested.

After each individual routine, and each functional element had been tested, the elements were combined and coupled into the simulation. Although tedious, I could not devise a less time consuming method than to exhaustively test each combination of maneuvers. Admittedly I did not test every parameter possible, but I did ensure that the order of inputs for the maneuvers had no impact on the results.

After the MIRAGE system was verified to my satisfaction, it was ready for testing with other users. These tests were planned to uncover idiosyncrasies or awkward requirements of the user in operating the simulation. The test group included several analysts and programmers from the Avionics Laboratory staff, the thesis advisor and a few other AFIT faculty members, several of my classmates in the GCS curriculum, my wife, and my six year old daughter! On the average, it seems to take about ten minutes of practice for a new user to fully comprehend all aspects of operating the simulation. The first few users pointed out some uncomfortable requirements, which I recoded to make a more friendly interface. The latest few seemed to sit down and run it like they had seen it before. This, I feel, is the ultimate test of interactive software.

With this test completed, and after several hours of operation with various users in control and no obvious discrepancies noted, I believe the software has been sufficiently tested and adequately verified.

VI Recommendations/Conclusion

Recommendations

To design, build, and test an interactive aircraft simulation with accompanying graphics interface, generate complete documentation for the software to [illegible] in the maintainable system, and document the entire project in a thesis report is a time consuming task. The general recommendations for this project were evident from it's conception; only the specifics were not available. It was known from the onset that the entire simulation would not be completed. My first recommendation, therefore, is to finish implementing the design. This requires interfacing with [illegible] routines to simulate EC maneuvers through the following two tasks:

a) feed the jamming factor associated with deploying a chaff pod to the [illegible] radar handling routines

b) feed the jammer information to the [illegible] radar handling routines

Two other tasks should be performed to totally implement my proposed design:

a) obtain the routine which converts X, Y aircraft position to latitude and longitude from the Avionics Laboratory and install it

b) implement the preplanned flight path [illegible]

mentioned in the Trouble Description section
of Chapter 1.

My second recommendation is also linked to earlier in this report: enhance the simulation. All states of charge throughout the [illegible] are either normal or failed. Some shading between these extremes, as well as small [illegible] [illegible] normal would make it a more realistic simulation. The symbols displayed on both the [illegible] visual display were simply characters from the ASCII set. Although functional, this is a bit cryptic. A separate graphic symbol could be drawn for each type of site, or received radar signal, giving the user a better picture of the environment. Since enhancement is limited only by imagination, I won't try to build a long list of possibilities here.

Conclusion

The development of the [illegible] software system, by all available measures, appears to have been successful. From my observations of novice user's first attempts at operating the system, I can honestly say that it is very "user friendly." The [illegible] structure, well documented modular design will [illegible] modification and enhancement [illegible] [illegible] of future [illegible]. [illegible] this time. The [illegible] [illegible] [illegible] novice [illegible] [illegible] [illegible] [illegible] [illegible] be processed and maintain the "real time" aspects of the simulation. The "build" function is currently held in processing for almost eighteen seconds of the eighty second [illegible] of [illegible] interaction time [illegible]. [illegible] for "build", the display and half for processing, while [illegible] for [illegible]. [illegible] available, that is 16 x 16 = 256 sites for use. [illegible] can process simultaneously [illegible] sites [illegible] will have to wait for complete implementation at the [illegible] Laboratory.

BIBLIOGRAPHY

1. McQuay, William K. "Proposed AFIT Thesis Topic," Electrical Engineering Department Proposed Thesis Topics, Item 25.

2. TEKTRONIX. PLOT-10 TERMINAL CONTROL USER'S MANUAL. Beaverton, Oregon: TEKTRONIX, Inc., 1974.

3. George, James E., Chairman. "Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH," Computer Graphics, 13(3), (August 1979).

4. Shannon, Robert E. System Simulation the Art and Science. Englewood Cliffs, N.J.: Prentice-Hall, Inc. 1975.

5. Krause, Randy, H. Design of an Interactive Graphics Input/Output System for an Aircraft Flight Simulation. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Instute of Technology, December, 1978.

6. McQuay, William K. Computer Simulation Methods for Military Operations Research. AFAL-TR-73-341. Wright-Patterson AFB, Ohio: Air Force Avionics Laboratory, October, 1973.

7. Morris, W. T. "On the Art of Modeling," Management Science, Vol 13, No 12, Aug 67.

8. Naylor, Thomas H. The Design of Computer Simulation Experiments. Durham, N.C.: Duke University Press, 1969.

9. 9022-78R. An Introduction to SADT, Structured Analysis and Design Technique. Waltham, Massachusetts: SofTech Inc., November 1976.

10. 9022-73.2 Structured Analysis Readers Guide. Waltham, Massachusetts: SofTech, Inc., May 1975.

11. Yourdon, Edward and Larry L. Constantine. Structured Design. New York, New York: Yourdon, Inc., 1975.

12. Weinberg, Victor. Structured Analysis. New York, New York: Yourdon, Inc., 1980.

13. Kernighan, Brian W. and P. J. Plauger. The Elements of Programming Style. New York, New York: Yourdon, Inc., 1978.

14. Electronic Warfare Principles. AFP51-3. Department of the Air Force, 1 September, 1978.

Appendix I

Equations

This appendix discusses two aspects of the mathematics used in the MIRAGE system which are not readily apparent either from reading the body of this thesis, or from studying the code. The first section of the Appendix discusses the derivation of the signal strength of a radar signal received at the aircraft which was used in constructing the Radar Warning Receiver Display. (Recall that the RWR displays the relative bearing of sites with respect to the aircraft, and the strength of the received radar signal.) The second section gives a brief error analysis for the approximations used throughout the simulation.

Radar Equations

The AADEM model presets all aircraft parameters and observes maneuver success from the ground environment's viewpoint. The MIRAGE system, on the other hand, interactively adjusts the aircraft parameters, and displays data from the aircraft's viewpoint. This required that some radar signal parameters used by the AADEM model be "backed up" to the aircraft perspective.

In order to construct the Radar Warning Receiver, the echo pulse power received at the aircraft of the radar from each site was needed.

The equation for this is:

$$S_{ac} = (P_t * G_t * \text{Loss})/4\pi r^2 .$$

where:

S_{ac} = echo pulse power (signal strength) received at the aircraft

P_t = peak power of the transmitted signal

G_t = gain of the antenna in the direction of the target

r = slant range between radar site and aircraft

[Ref 14:2.6-2.7]

However, since the AADEM model uses the site's perspective, it deals with signal strength reflected to the site. This equation is similar, but is affected by the cross-section area of the aircraft, twice the distance, and the affective area of the receiving antenna.

Specifically,

$$S_{site} =((P_t*G_t*\text{loss})/4\pi r^2)*((\text{Xsec}*\text{loss}'*A_r)/4\pi r^2)$$

$$S_{site} =S_{ac}*(\text{Xsec}/4\pi r^2)*\text{loss}'*A_r$$

$$\text{or } S_{ac} =S_{site}*((4\pi r^2/(\text{Xsec}*\text{loss}'*A_r))$$

where:

A_r = area of the receiving antenna

An equation was needed which determined the one way loss, and "unfigured" the affect of aircraft cross-section and the receiver antenna. In subroutine STRNTH, RLOSS is a computation which determines the one-way loss needed from the two-way loss available in AADEM. FACTOR, then, is the solution to the equation discussed above expressed in terms of the AADEM variables and RLOSS. Precise definitions for the AADEM variables used are available in the AADEM documentation.

Error Analysis

Although this section in entitled error analysis, it should be emphasized that the MIRAGE system's equations are exact except when the aircraft's speed or direction are changing. In comparsion to its total mission time, this should be a relatively small percentage. For simplicity, consider the two cases disjoint. The worst case performance will be no more than the sum of the two errors.

First look at the case of constant heading, with a constant acceleration or deceleration. Also, assume that our velocity is totally in the X direction.

We know that velocity = $V = dx/dt$

and acceleration = $a = dV/dt$.

Then $V(t) = V_o + \int_0^t a\, dt = V_o + at$

In the MIRAGE system a is constant and for any time increment, Δt,

$$X = X_o + \int_0^{\Delta t} (V_o + at)\, dt$$

$$\Delta x = V_o\, t + (1/2)a\, t^2 \Big|_0^{\Delta t} \text{ when acceleration is constant}$$

$$\Delta x = V_o\, \Delta t + (1/2)\, a\, (\Delta t)^2$$

The MIRAGE system approximates using $\Delta x = V_o \Delta t$

Thus the error = $(1/2)\, a\, \Delta t^2$

and % error $= ((1/2)\, a \Delta t^2)/(V_o \Delta t) * 100$

$= [(1/2)\, a \Delta t/V_o] * 100$

At a typical cruise airspeed of 400K using the maximum acceleration of 15K/sec and Δt of 1 second as used in MIRAGE

% error = [(1/2) * (15) * (1)/400] * 100 = 1.875%

For the second case, consider constant airspeed, but a turn. We will consider a standard X, Y coordinate system with θ measured positive clockwise from the +Y axis.

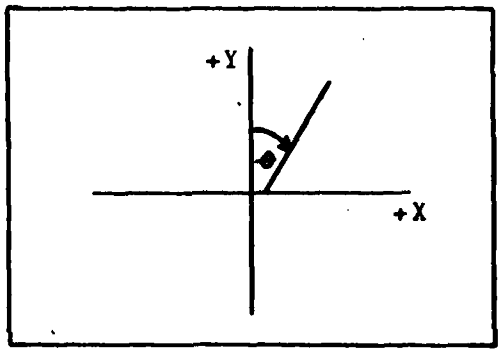


FIGURE 13
Theta Measurement

Then the X component of the velocity $V_x = V \sin\theta$

Let w = turn rate = $d\theta/dt$

and $\Delta x = V \int_0^{\Delta t} \sin\theta(t)\,dt$ where $\theta(t) = \theta_0 + wt$,

$\Delta x = V \int_0^{\Delta t} \sin(\theta_0 + wt)\,dt$

Let $\phi = \theta_0 + wt$ then $d\phi = w\,dt \rightarrow dt = d\phi/w$

then $\Delta x = V \int_0^{\Delta t} \sin(\phi)\, w\, d\phi$

$= (V/w)[-\cos\phi]\Big|_0^{\Delta t}$

$= (-V/w)[\cos(\theta_0 + wt)]\Big|_0^{\Delta t}$

$= (-V/w)[\cos(\theta_0 + w\Delta t) - \cos\theta_0]$

For example, let $\theta_0 = \pi/2$ then

$$\Delta x = (-V/w)\ [\mathrm{Cos}(\pi/2 + w\Delta t)]$$

$$= (-V/w)\ [\mathrm{Cos}\ \pi/2\ \mathrm{Cos}\ w\Delta t - \mathrm{Sin}\ \pi/2\ \mathrm{Sin}\ w\Delta t]$$

$$\Delta x = (-V/w)\ [-\mathrm{Sin}\ w\Delta t]$$

We know the Maclaurin series of

$$\mathrm{Sin}\ X = X - (X^3/3!) + (X^5/5!) - (X_7/7!) \ldots$$

so $\Delta x \equiv (V/w)\ [(w\Delta t) - (w\Delta t)^3/3!]$ using the first two terms of the series

$$\Delta x \equiv V\Delta t - (Vw^2\Delta t^3)/6$$

The MIRAGE system uses $\Delta x \equiv V\Delta t$

which has error approximately $V\ w^2\Delta t^3/6$

and % error $= [((Vw^2\Delta t^3)/6)\ /\ V\Delta t\ ] * 100$

$$= [(w^2\Delta t^2)/6] * 100$$

In MIRAGE, $\Delta t = 1$ sec

maximum w = 18 degrees/sec =.3141592 radians/sec

Thus % error = $(.3141592)^2 * (1)^2 * 100\ /\ 6 = 1.64493\%$

Since % error grows as Δt^2 in the second case, clearly a Δt much greater than one would cause significantly larger errors. A smaller Δt would reduce the error, but I feel that these errors are tolerable as it stands. If a future user disagrees, it requires a simple edit in the data statement for "DT" in subroutine GINITL to reduce the % error as desired.

Appendix II

SADT Descriptions

This appendix contains the complete SADT description for the MIRAGE software system. Recall that one criterion for stopping the decomposition of a function is that the function can be accomplished by a known tool. Since the MIRAGE software system interacts with the AADEM model, several AADEM routines were used as tools. This explains why the following nodes/boxes were not decomposed further.

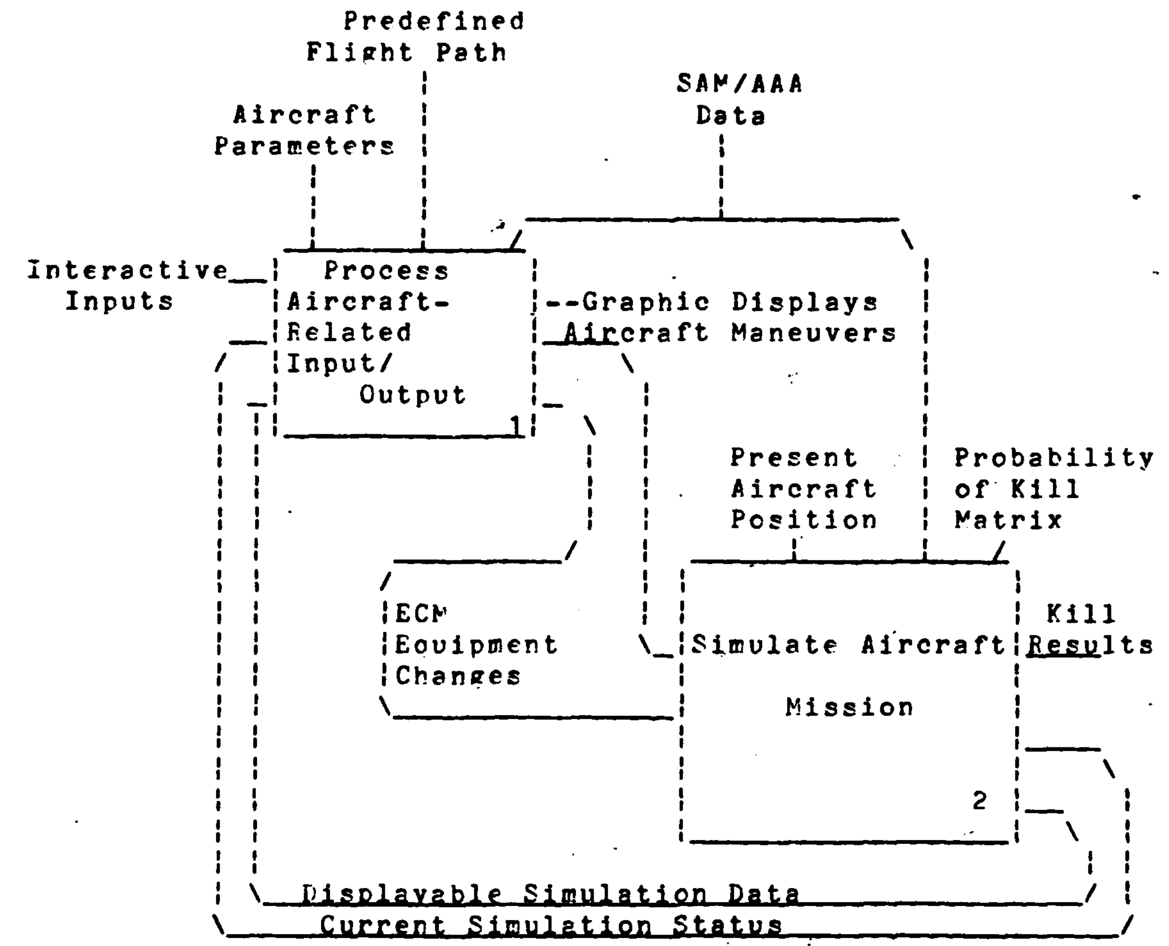
Node A125 box 1

Node A125 box 2

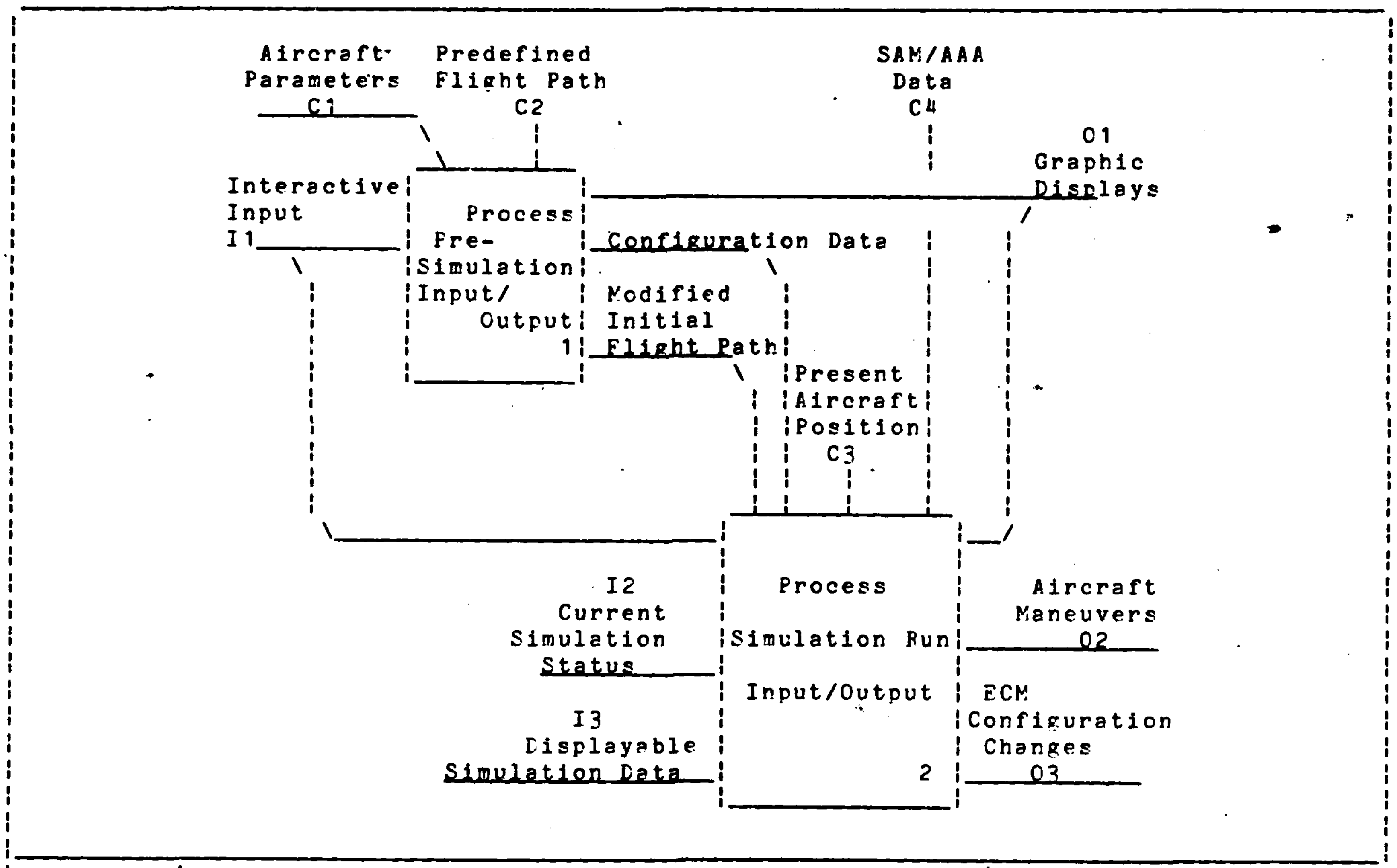
Node A22 box 3

Node A23 box 1

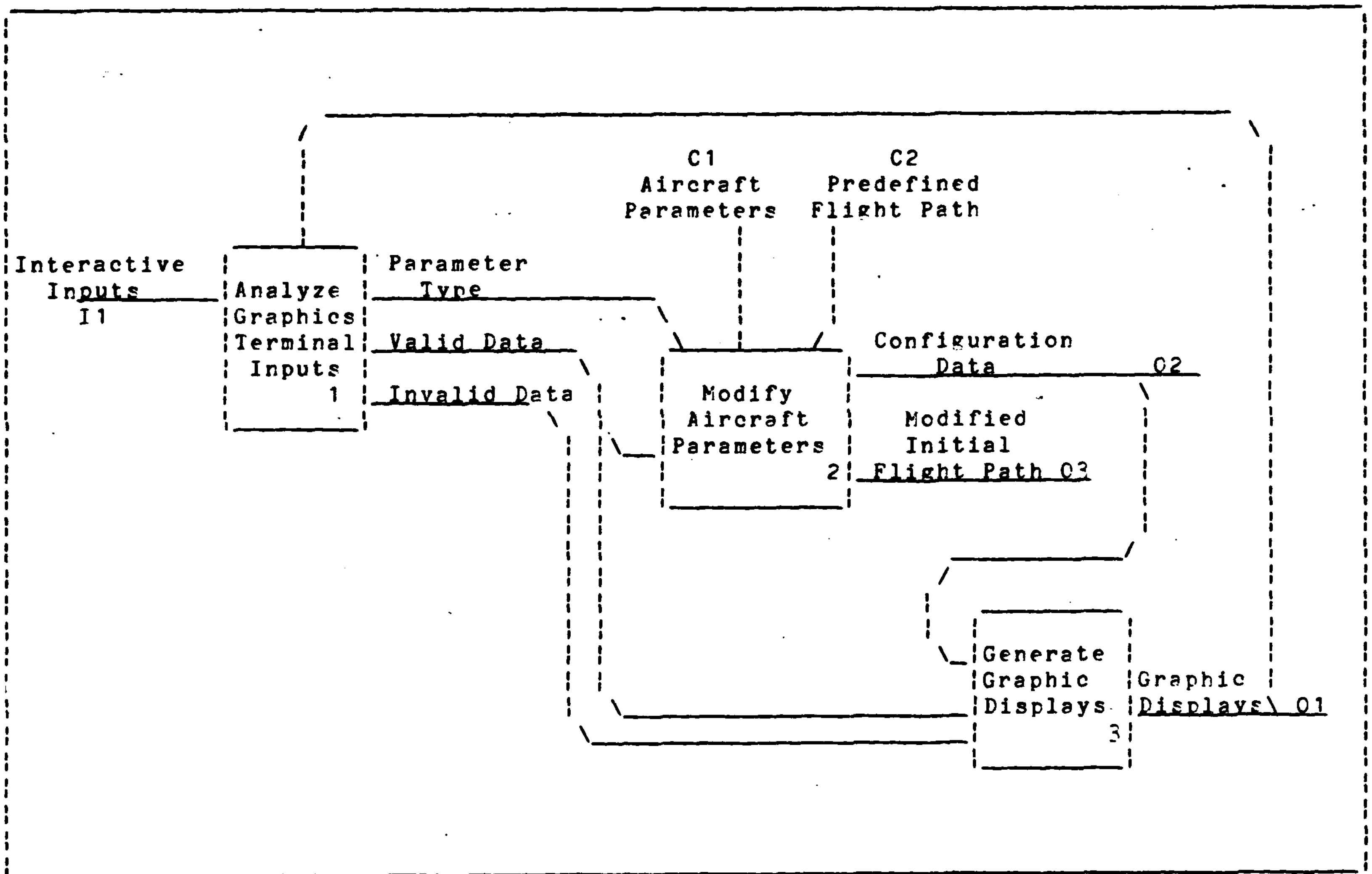
Node A23 box 2


Predefined Flight Path
Aircraft Parameters
SAM/AAA Data
Interactive Inputs
Process Aircraft-Related Input/ Output
1
--Graphic Displays
Aircraft Maneuvers
Present Aircraft Position
Probability of Kill Matrix
ECM Equipment Changes
Simulate Aircraft Mission
2
Kill Results
Displayable Simulation Data
Current Simulation Status

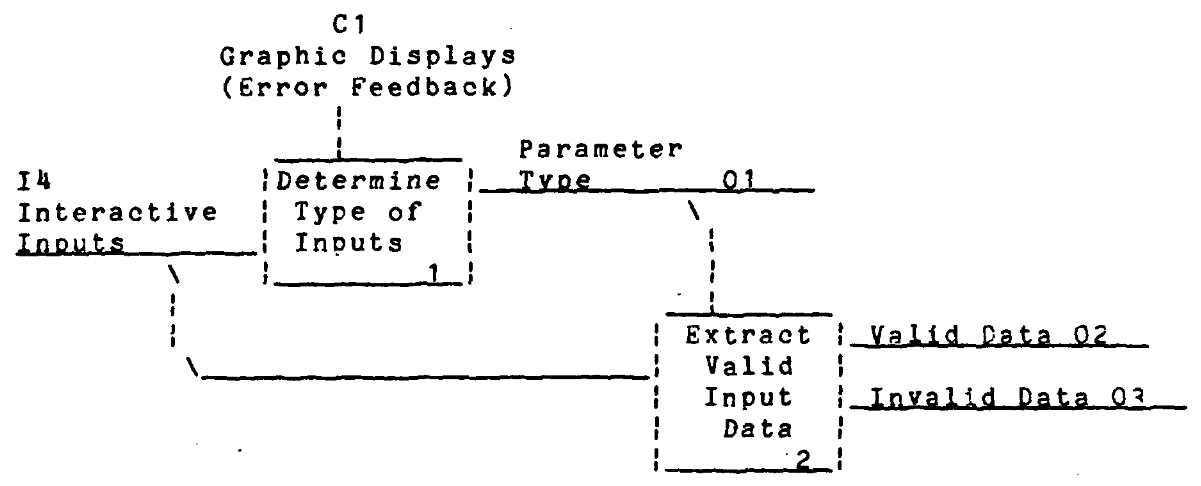
Node A0 | An Interactive Bombing Mission Simulation with Computer Graphics Interface


Aircraft Parameters C1
Predefined Flight Path C2
SAM/AAA Data C4
O1 Graphic Displays
Interactive Input I1
Process Pre-Simulation Input/Output 1
Configuration Data
Modified Initial Flight Path
Present Aircraft Position C3
I2 Current Simulation Status
Process Simulation Run Input/Output 2
Aircraft Maneuvers O2
I3 Displayable Simulation Data
ECM Configuration Changes O3

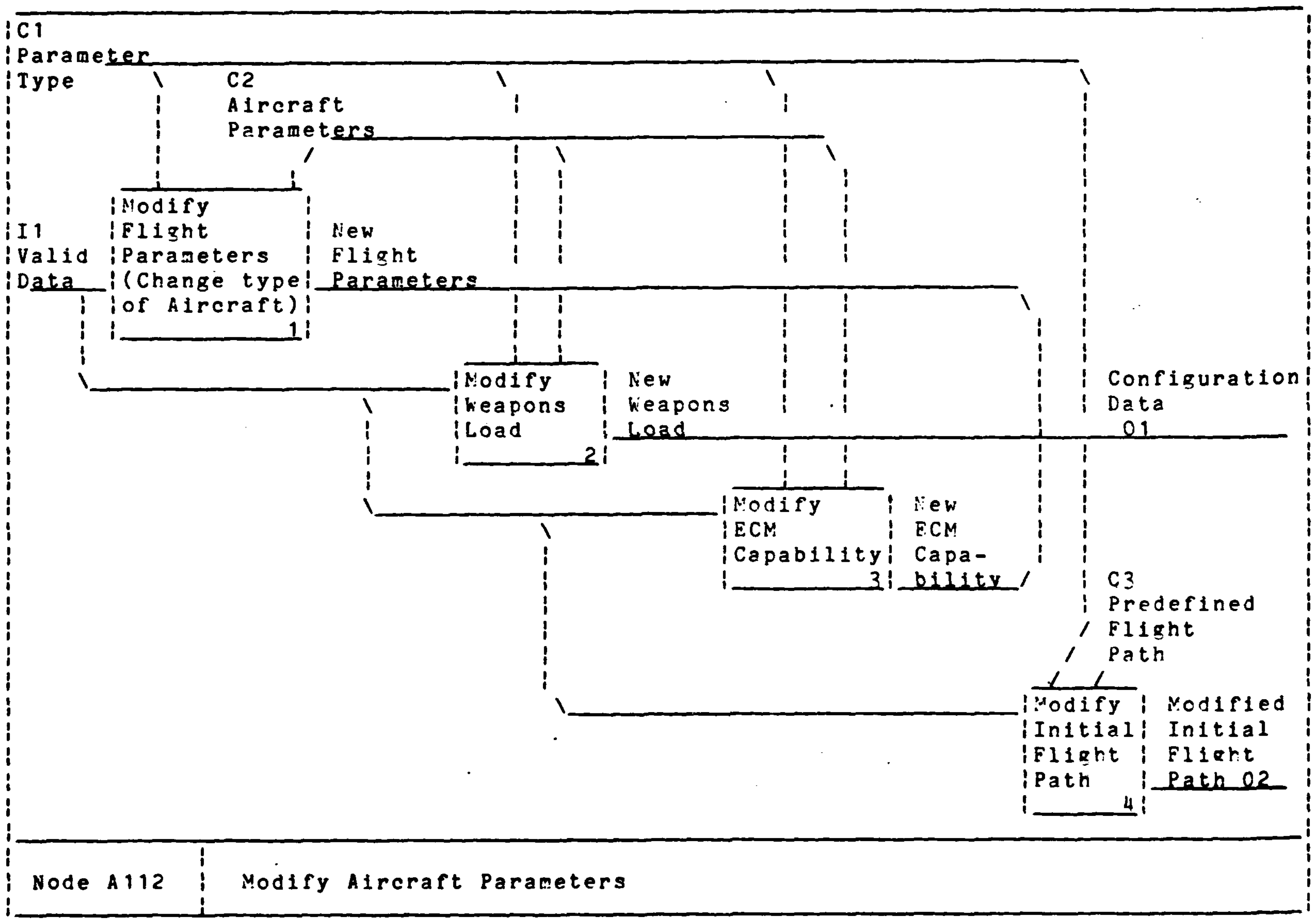
Node A1 | Process Aircraft Related Input/Output

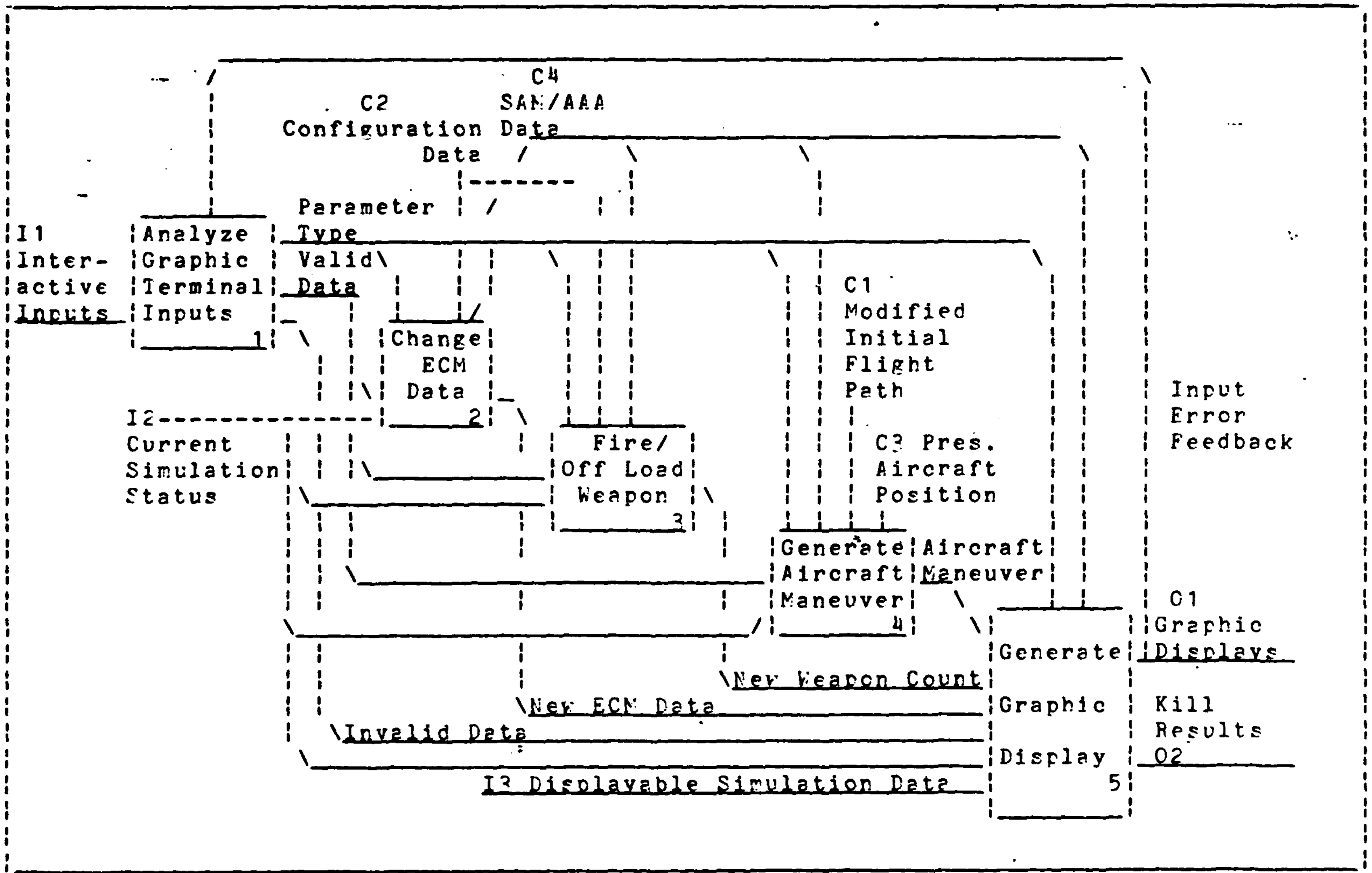


Node A11 Process Pre-simulation Input/Output

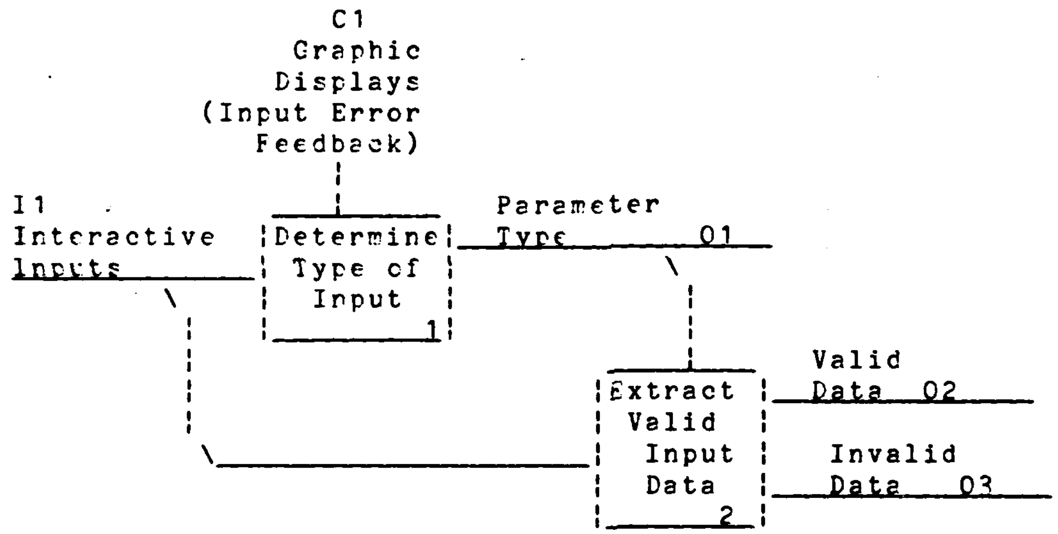


Node A111	Analyze Graphic Terminal Inputs

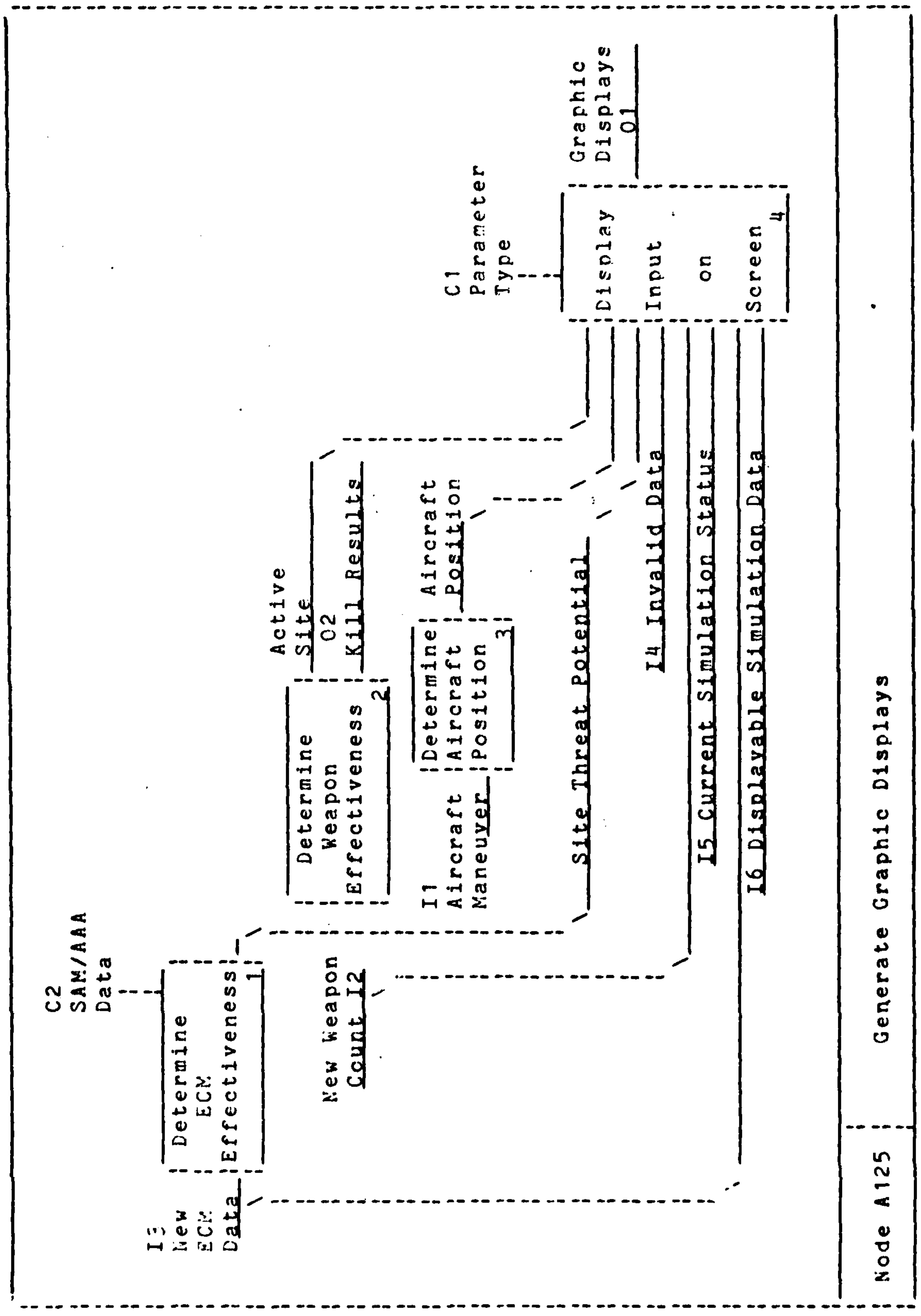

C1
Parameter
Type
C2
Aircraft
Parameters
I1
Valid
Data
Modify
Flight
Parameters
(Change type
of Aircraft)
1
New
Flight
Parameters
Modify
Weapons
Load
2
New
Weapons
Load
Configuration
Data
01
Modify
ECM
Capability
3
New
ECM
Capa-
bility
C3
Predefined
Flight
Path
Modify
Initial
Flight
Path
4
Modified
Initial
Flight
Path 02
Node A112
Modify Aircraft Parameters

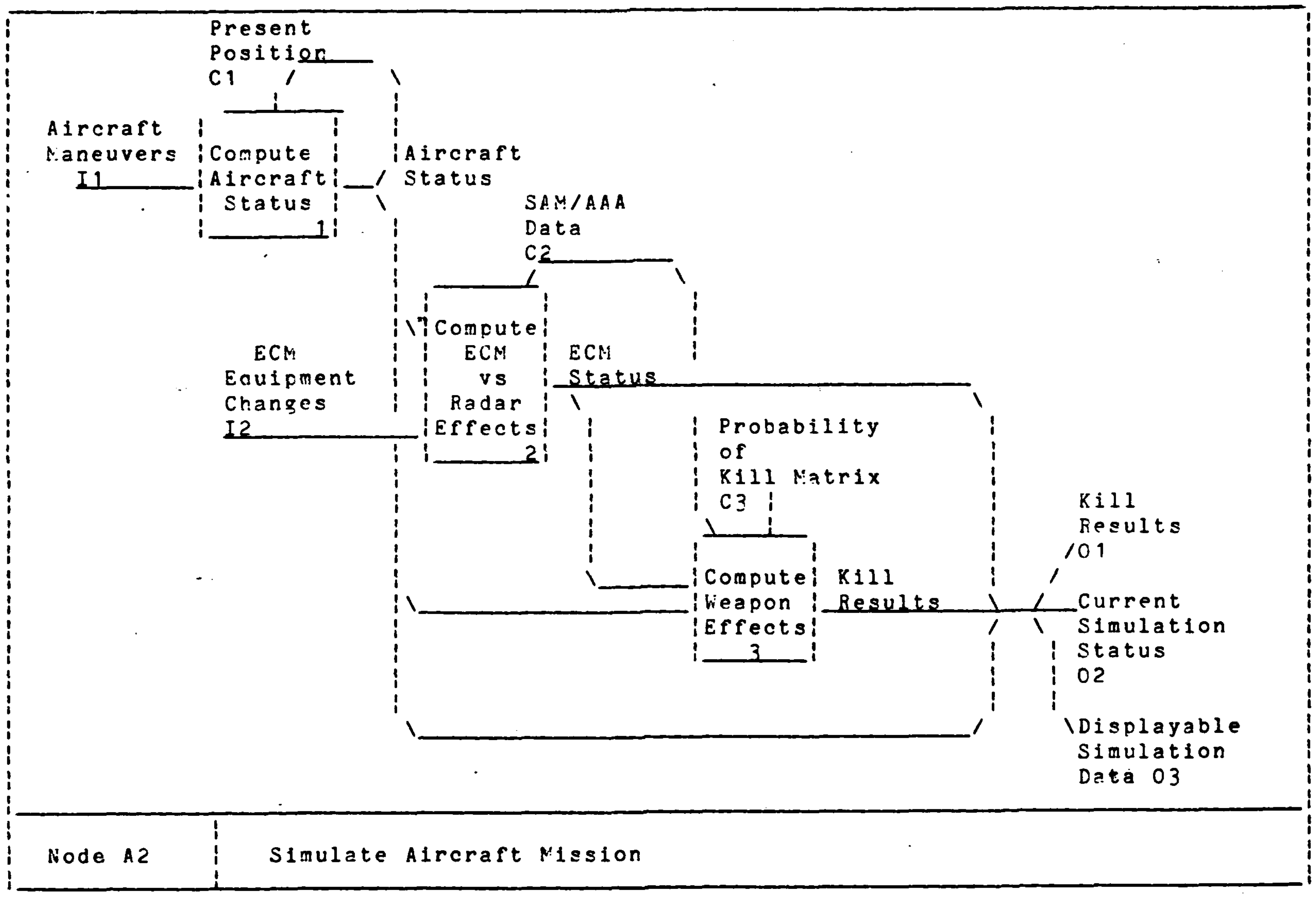


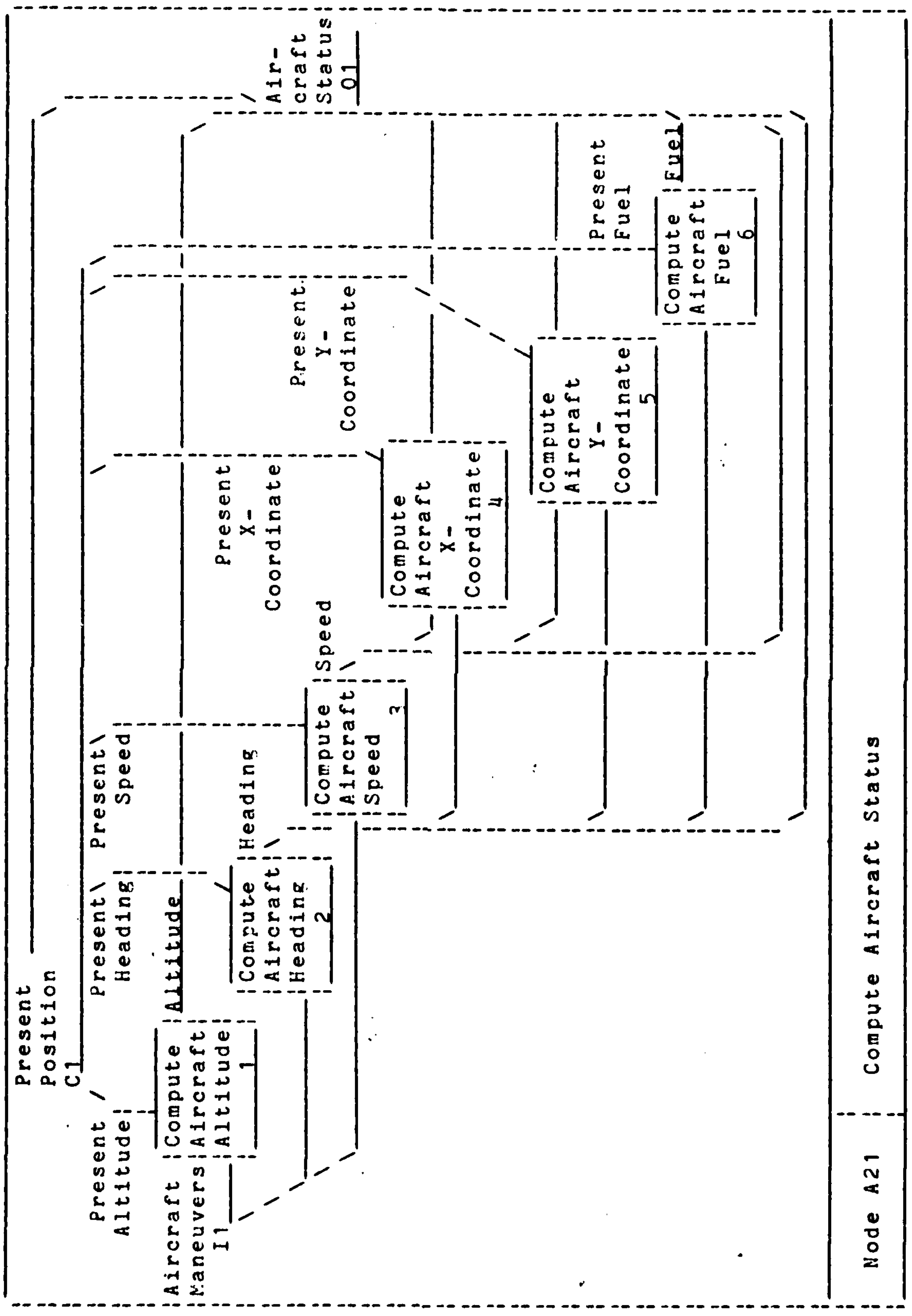
Node A12 | Process Simulation Run Input/Output

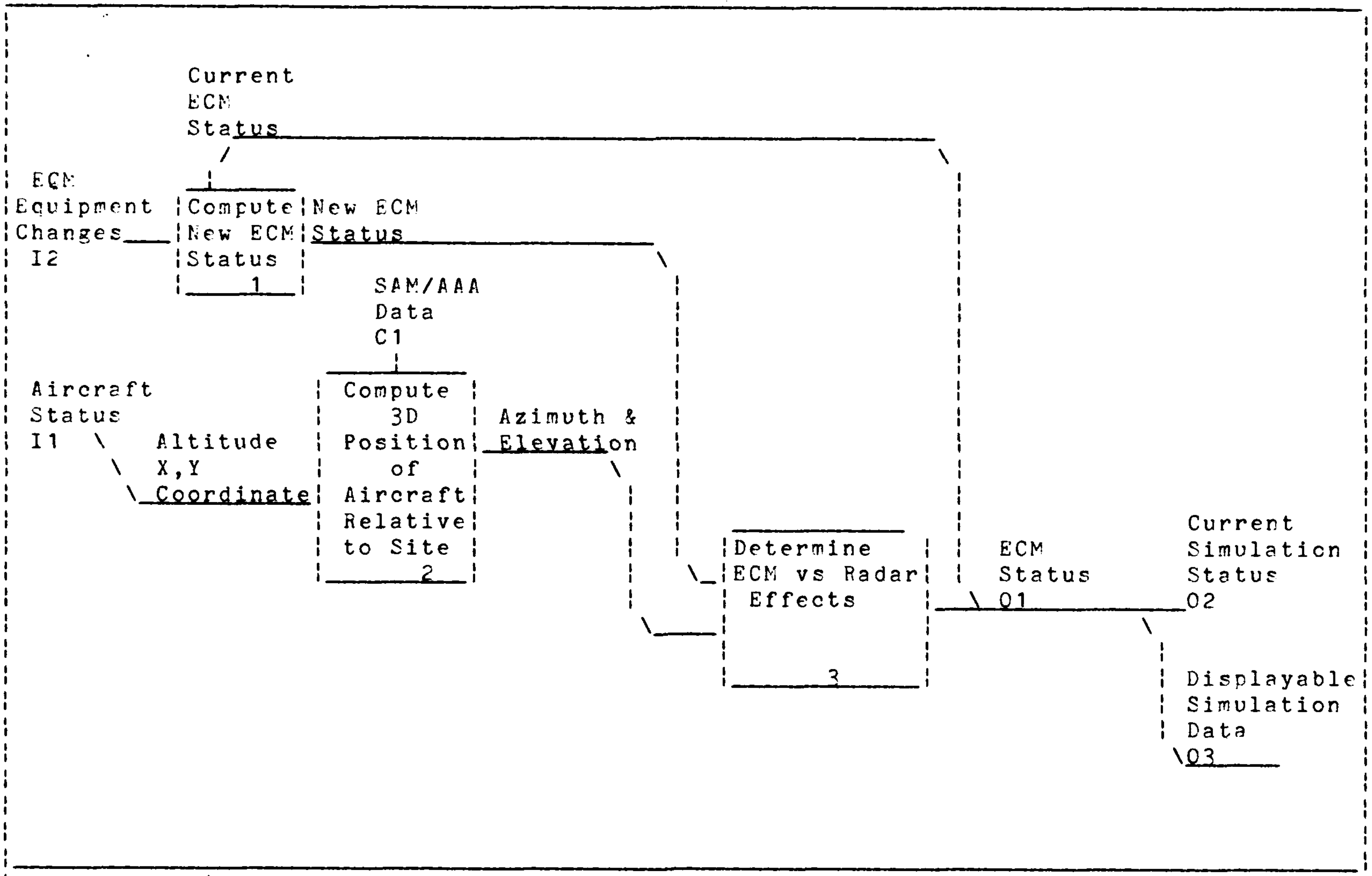


Node A121 | Analyze Graphic Terminal Inputs

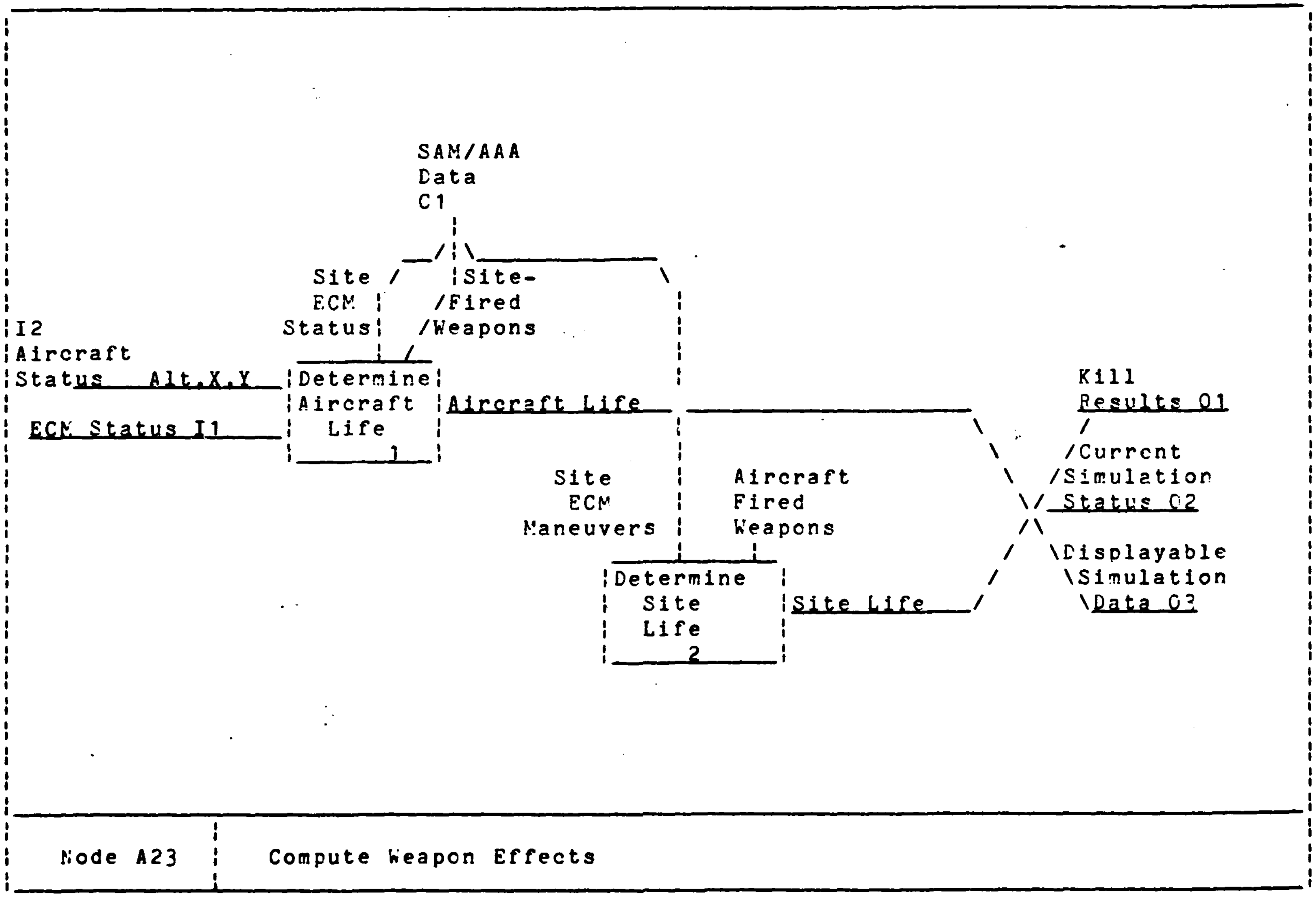

I3 New ECM Data
C2 SAM/AAA Data
Determine ECM Effectiveness 1
New Weapon Count I2
Determine Weapon Effectiveness 2
Active Site O2 Kill Results
I1 Aircraft Maneuver
Determine Aircraft Position 3
Aircraft Position
Site Threat Potential
I4 Invalid Data
I5 Current Simulation Status
I6 Displayable Simulation Data
C1 Parameter Type
Display Input on Screen 4
Graphic Displays O1
Node A125
Generate Graphic Displays


Present Position C1
Aircraft Maneuvers I1
Compute Aircraft Status 1
Aircraft Status
SAM/AAA Data C2
Compute ECM vs Radar Effects 2
ECM Equipment Changes I2
ECM Status
Probability of Kill Matrix C3
Compute Weapon Effects 3
Kill Results
Kill Results O1
Current Simulation Status O2
Displayable Simulation Data O3
Node A2
Simulate Aircraft Mission


Present Position C1
Present Heading
Present Speed
Air-craft Status O1
Present Altitude
Aircraft Maneuvers I1
Compute Aircraft Altitude 1
Altitude
Compute Aircraft Heading 2
Heading
Compute Aircraft Speed 3
Speed
Present X-Coordinate
Compute Aircraft X-Coordinate 4
Present Y-Coordinate
Compute Aircraft Y-Coordinate 5
Present Fuel
Compute Aircraft Fuel 6
Fuel
Node A21
Compute Aircraft Status



Node A22 | Compute ECM vs Radar Effects


SAM/AAA Data C1
Site ECM Status
Site-Fired Weapons
I2 Aircraft Status
Alt.X.Y
ECM Status I1
Determine Aircraft Life 1
Aircraft Life
Site ECM Maneuvers
Aircraft Fired Weapons
Determine Site Life 2
Site Life
Kill Results O1
Current Simulation Status O2
Displayable Simulation Data O3
Node A23
Compute Weapon Effects

[illegible]

```
NAME:        [illegible]
TYPE:        [illegible]
CALLS:       [illegible], [illegible], [illegible], [illegible]
CALLED BY:   [illegible]

NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine CRITL, AICPT
PURPOSE:     Abort altitude

NAME:        [illegible]
TYPE:        Common block name
USE:         Subroutine CRITL, AICPT
PURPOSE:     Parameters for aborting the mission

NAME:        [illegible]
TYPE:        Common variable (AICPT)
USE:         Subroutine CRITL, AICPT
PURPOSE:     Abort heading

NAME:        AICPT
TYPE:        Subroutine name
PURPOSE:     Responds to pilot termination of [illegible]
CALLS:       LVLOFF, [illegible], HOLDPT, [illegible], [illegible], [illegible],
             [illegible], [illegible], [illegible]
CALLED BY:   FLYTO

NAME:        [illegible]
TYPE:        Common variable (AICPT)
USE:         Subroutine CRITL, AICPT
PURPOSE:     Abort speed

NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine [illegible], [illegible], CRITL
PURPOSE:     Fuel use rate adjustment for acceleration

NAME:        [illegible]
TYPE:        Subroutine [illegible]
PURPOSE:     [illegible] pilot [illegible]
             [illegible]
CALLS:       [illegible], [illegible], [illegible]
CALLED BY:   [illegible], FLYTO

NAME:        [illegible]
TYPE:        [illegible]
USE:         [illegible]
PURPOSE:     [illegible]
             [illegible]
```

Reproduced from best available copy.

[illegible] : [illegible]
[illegible] : [illegible] (ABCD [illegible])
[illegible] : Subroutine [illegible], [illegible]
[illegible]: [illegible] adjustment for [illegible] [illegible]

[illegible] : [illegible]
[illegible] : [illegible] (ABCD [illegible])
[illegible] : Subroutine [illegible], [illegible]
[illegible]: [illegible] adjustment for [illegible]
[illegible]

[illegible] : [illegible]
[illegible] : [illegible]
[illegible] : Subroutine [illegible], [illegible]
[illegible] : [illegible] parameters

[illegible] : [illegible]
[illegible] : Common [illegible]
[illegible] : Subroutine [illegible], [illegible], [illegible], [illegible], [illegible],
TUR [illegible], [illegible], [illegible], [illegible], [illegible], [illegible],
[illegible]
[illegible]: All [illegible] information relating to aircraft flight status

[illegible] : [illegible]
Type : Common variable (LOCAT [illegible])
Use: Subroutine [illegible], STATVL, FLTSIT, [illegible],
[illegible]
Purpose: X-coordinate of current aircraft position

[illegible] : [illegible]
Type: Common variable (LOCAT)
Use: Subroutine [illegible], STATVL, FLTSIT, [illegible],
[illegible]
Purpose : Y-coordinate of current aircraft position

[illegible] : [illegible]
[illegible] : [illegible] subroutine
[illegible] : [illegible] documentation

[illegible] : [illegible]
[illegible] : [illegible] subroutine
[illegible] : [illegible] documentation

[illegible] : [illegible]
[illegible] : [illegible] ([illegible])
[illegible] : Subroutine [illegible], [illegible], [illegible], [illegible],
[illegible], [illegible], [illegible]
[illegible]: Current aircraft [illegible]

[illegible]: [illegible]
Type: Local variable
Use: Subroutine CIRCLE
Purpose: x-coordinate, in virtual units, of the center of the circle

[illegible]: [illegible]
Type: Local variable
Use: Subroutine CIRCLE
Purpose: y-coordinate, in virtual units, of the center of the circle

[illegible]: [illegible]
Type: Common variable ([illegible])
Use: Subroutine [illegible], [illegible], [illegible]
Purpose: Probability of kill [illegible]

[illegible]: [illegible]
Type: Subroutine [illegible]
Purpose: [illegible]
from the [illegible]
Calls: [illegible]
Called by: [illegible]
[illegible]

[illegible]: [illegible]
Type: Common variable ([illegible])
Use: Subroutine [illegible], [illegible]
Purpose: [illegible]

[illegible]: [illegible]
Type: Subroutine [illegible]
Purpose: Responds to the pilot [illegible] button
[illegible]: [illegible]
[illegible]: [illegible]

[illegible]: [illegible]
Type: Common variable ([illegible])
Use: Subroutine [illegible], [illegible]
[illegible]: [illegible]

[illegible]: [illegible]
Type: [illegible]
Purpose: [illegible]

[illegible]: [illegible]
Type: Subroutine [illegible]
Purpose: [illegible]
the [illegible]
[illegible]: [illegible]
[illegible]: [illegible]

NAME: CLIM[illegible]
TYPE: Local variable
USE: Subroutine CLI[illegible], LVLOFF
PURPOSE: Rate to level [illegible] level off [illegible] the climb

NAME: CLIMB
TYPE: Subroutine name
PURPOSE: Simulates pilot actions to cause the aircraft to climb
CALLS: [illegible], [illegible], [illegible]
CALLED BY: [illegible], [illegible]

NAME: [illegible]
TYPE: Common block name
USE: Subroutine CLIMB, [illegible], [illegible]
PURPOSE: Simulated hours, minutes, seconds

NAME: CONT
TYPE: Local variable
USE: Subroutine [illegible], [illegible], [illegible]
PURPOSE: Character input by user to cause the routine to continue operation

NAME: COST
TYPE: Local variable
USE: Subroutine CIRCLE
PURPOSE: Cosine of THETA

NAME: CRATE
TYPE: Local variable
USE: Subroutine FLYAC, CLIMB
PURPOSE: Climb rate parameter

NAME: CTRLC
TYPE: Subroutine name
PURPOSE: Sets flags to indicate that the user wants to change inputs
CALLS: [illegible]
CALLED BY: [illegible]

NAME: CX
TYPE: Local variable
USE: Subroutine CIRCLE
PURPOSE: x-coordinate [illegible] points [illegible] by the circle [illegible]

NAME: CY
TYPE: Local variable
USE: Subroutine CIRCLE
PURPOSE: y-coordinate [illegible] points [illegible] by the circle [illegible]

```
NAME:        DCURSR
TYPE:        TEKTRONIX subroutine
PURPOSE:     See TEKTRONIX documentation

NAME:        DECEL
TYPE:        Subroutine name
PURPOSE:     Simulates pilot motions to cause the aircraft
             to decelerate
CALLS:       SETPP, [illegible], [illegible]
CALLED BY:   [illegible], FLYTO

NAME:        DESCNT
TYPE:        Subroutine name
PURPOSE:     Simulates pilot motions to cause the aircraft
             to descend
CALLS:       SETPP, [illegible], LVLOFF
CALLED BY:   [illegible], FLYTO

NAME:        [illegible]
TYPE:        Local variable
USE:         Subroutine DESCNT
PURPOSE:     Flag to LVLOFF indicating level off from a
             descent

NAME:        DIFF
TYPE:        Local variable
USE:         Subroutine PIKSIT
PURPOSE:     Slant distance from cursor to site

NAME:        DISP
TYPE:        Common block name
USE:         Subroutines FLYTO, [illegible], [illegible], [illegible], [illegible],
             [illegible], [illegible], PIKSIT, [illegible]
PURPOSE:     Indicates whether the VISUAL or RADAR display
             should be presented

NAME:        DISPLY
TYPE:        Subroutine name
PURPOSE:     Draws the interactive display
CALLS:       [illegible], [illegible], [illegible], [illegible]
CALLED BY:   [illegible]

NAME:        DIST
TYPE:        Local variable
USE:         Subroutines [illegible], [illegible]
PURPOSE:     Distance from cursor pick to site or aircraft
             to site

NAME:        [illegible]
TYPE:        Local variable
USE:         Subroutine [illegible]
PURPOSE:     Slant range distance ([illegible]) between
             site and aircraft
```

```
NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine [illegible], LVL[illegible], [illegible]
PURPOSE:     [illegible] use rate adjust ent for [illegible] descent

NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine [illegible]TL, [illegible], [illegible]
PURPOSE:     True when aircraft is [illegible]

NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine [illegible], [illegible]
PURPOSE:     [illegible] use rate adjustment for [illegible] rate
             descent

NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine [illegible]
PURPOSE:     [illegible] use rate adjustment for [illegible] rate
             descent

NAME:        [illegible]
TYPE:        Local variable
USE:         Subroutine [illegible], [illegible]
PURPOSE:     Descent rate parameter

NAME:        [illegible]
TYPE:        T[illegible] subroutine
PURPOSE:     See T[illegible] documentation

NAME:        [illegible]
TYPE:        Subroutine name
PURPOSE:     Does all of the static line drawing for the
             man-in-loop display
CALLS:       [illegible], [illegible], [illegible], [illegible], [illegible]
CALLED BY:   [illegible]

NAME:        [illegible]
TYPE:        T[illegible] subroutine
PURPOSE:     See T[illegible] documentation

NAME:        [illegible]
TYPE:        T[illegible] subroutine
PURPOSE:     See T[illegible] documentation

NAME:        [illegible]
TYPE:        Common variable ([illegible])
USE:         Subroutine [illegible], [illegible]
PURPOSE:     [illegible]

NAME:        [illegible]
TYPE:        [illegible]
PURPOSE:     [illegible]
```

NAME: [illegible]
TYPE: Common block name
USE: Subroutine [illegible], [illegible], [illegible]
PURPOSE: [illegible] for [illegible]

NAME: [illegible]
TYPE: Subroutine name
PURPOSE: Writes the values for the [illegible]
CALLS: [illegible], [illegible], [illegible], [illegible]
CALLED BY: [illegible]

NAME: [illegible]
TYPE: Subroutine name
PURPOSE: Writes the [illegible] information for the "[illegible] [illegible]" block
CALLS: [illegible], [illegible], [illegible], [illegible]
CALLED BY: [illegible]

NAME: [illegible]
TYPE: Subroutine name
PURPOSE: Enables interrupts
CALLS: System level routines
CALLED BY: [illegible], [illegible]

NAME: [illegible]
TYPE: Subroutine name
PURPOSE: Terminates TEKTRONIX graphics mode and clears the screen
CALLS: [illegible], [illegible], [illegible], [illegible]
CALLED BY: [illegible]

NAME: FACTOR
TYPE: Local variable
USE: Subroutine [illegible]
PURPOSE: Multiplier used to "[illegible]" [illegible] variables to recover one-way signal strength

NAME: FACT
TYPE: Local variable
USE: Subroutine [illegible]
PURPOSE: [illegible] acceleration

NAME: [illegible]
TYPE: Common variable ([illegible])
USE: Subroutine [illegible], [illegible]
PURPOSE: [illegible]

NAME: [illegible]
TYPE: Common variable ([illegible])
USE: Subroutine [illegible], [illegible]
PURPOSE: [illegible]

NAME: FFMAX
TYPE: [illegible] (FUELFL)
USE: Subroutine GINITL, SETFF
PURPOSE: Average fuel use rate, full throttle, at flight altitude

NAME: FLAG
TYPE: Local variable
USE: Subroutine LVLOFF, SETSPD
PURPOSE: Indicates whether from a climb or a descent, acceleration or deceleration

NAME: FLARE
TYPE: Subroutine name
PURPOSE: Responds to the pilot picking the "F" button
CALLS:
CALLED BY: FLYAC

NAME: FLOWCH
TYPE: Local variable
USE: Subroutine SETFF, RSETFF
PURPOSE: Input parameter - amount to change fuel use rate

NAME: FLPRAM
TYPE: Common block name
USE: Subroutine GINITL, DESCND, CLIMB, DECEL, ACCEL, TURN
PURPOSE: Fuel use rate adjustment for transient flight states

NAME: FLRDEG
TYPE: Common variable (ECMDEG)
USE: Subroutine GINITL, WRITER
PURPOSE: Degrade factor for flare

NAME: FLYAC
TYPE: Subroutine name
PURPOSE: Responds to the pilot's menu picks
Prompts pilot for information as required
Sets simulation status flag to indicate that the aircraft is in a transient state
Calls appropriate subroutines to adjust the aircraft position accordingly
CALLS: DCURSR, REFRSH, MOVABS, ANSTR, ANMODE, HELP, CHAFF, FLARE, ABORT, RESUME, WEAPON, JAMMER, LVLOFF, DESCND, CLIMB, SETSPD, DECEL, ACCEL, ROLOUT, TURN, ENDG
CALLED BY: PROCES

NAME: FREQ (5)
TYPE: Common variable (JAMM)
USE: Subroutine GINITL, INPUT, ECMVAL, JAMMER
PURPOSE: Frequency setting of respective jammer

NAME:	FRPISQ
TYPE:	Local variable
USE:	Subroutine STRNTH
PURPOSE:	Constant - $(4*Pi)^2$

NAME:	FSTING
TYPE:	Common variable (STATUS)
USE:	Subroutine GINITL, FLYAC, STATVL
PURPOSE:	TRUE when aircraft is accelerating

NAME:	FUELS
TYPE:	Common block name
USE:	Subroutine GINITL, SETFF, RSETFF
PURPOSE:	All global information relating to aircraft fuel

NAME:	FULFLO
TYPE:	Common block name
USE:	Subroutine GINITL, SETFF, RSETFF, WEAPON
PURPOSE:	Aircraft fuel use rate parameters

NAME:	FULRAT
TYPE:	Common variable (FUELS)
USE:	Subroutine GINITL, SETFF, RSETFF
PURPOSE:	Fuel use rate

NAME:	FULTOT
TYPE:	Common variable
USE:	Subroutine GINITL, STATVL
PURPOSE:	Total fuel remaining

NAME:	GETTGT
TYPE:	Subroutine name
PURPOSE:	Plots enemy sites for the visual display and gets user pick for target
CALLS:	PLTSIT, PIKSIT
CALLED BY:	WEAPON

NAME:	GINITL
TYPE:	Subroutine name
PURPOSE:	Initializes all global variables used in the simulation.
CALLS:	START
CALLED BY:	PRELIM

NAME:	GSETUP
TYPE:	Subroutine name
PURPOSE:	Refreshes the screen and calls for pilot inputs
CALLS:	REFRSH, CHRSIZ, ANMODE, MOVABS, ANSTR
CALLED BY:	PROCES, PRELIM, JAMMER

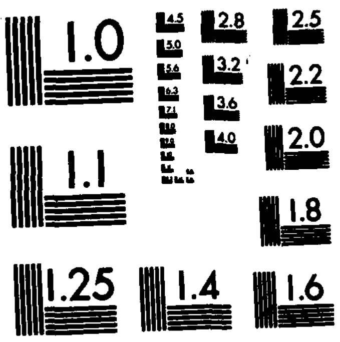
AD-A124 661 AN INTERACTIVE BOMBING MISSION SIMULATION WITH COMPUTER GRAPHICS INTERFACE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M J GOCI 2/2

UNCLASSIFIED DEC 82 AFIT/GCS/MA/82D-4 F/G 9/2 NL

END

FILMED

DTIC


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

NAME:	<u>GSTATUS</u>
TYPE:	Common block name
USE:	Subroutine SETDEV, ENDG
PURPOSE:	All global graphics information
NAME:	<u>HDG</u>
TYPE:	Common variable (ACSTAT)
USE:	Subroutine TURN, ROLOUT, GINITL, PIKSIT, PLTSIT
PURPOSE:	Current heading of the aircraft
NAME:	<u>HELP</u>
TYPE:	Subroutine name
PURPOSE:	Prints information on how to operate the simulation
CALLS:	CHRSIZ, NEWPAG, ANMODE
CALLED BY:	FLYAC, GETTGT, INFO3, PIKSIT
NAME:	<u>HIDEIT</u>
TYPE:	Subroutine name
PURPOSE:	Hides the interupt character echo
CALLS:	CHRSIZ, MOVABS, ANMODE
CALLED BY:	PROCES
NAME:	<u>HITGND</u>
TYPE:	Subroutine name
PURPOSE:	Handles functions if the aircraft inadvertently is flown into the ground
CALLS:	NEWPAG, CHRSIZ, MOVABS, ANSTR, ANCHO, GSETUP, ANMODE, BOXER, WAIT, LVLOFF
CALLED BY:	STALL, STATVL
NAME:	<u>HSMSPK</u>
TYPE:	Common variable (WEPPKS)
USE:	Subroutine GINITL, INPUT, WEAPON
PURPOSE:	Probability of kill for an IR missile
NAME:	<u>HT</u>
TYPE:	Local constant
USE:	Subroutines WORDS
PURPOSE:	Y-coordinate in absolute units of the lowest line on which to write MAX in the menu
NAME:	<u>I (10)</u>
TYPE:	Local variable
USE:	Subroutine WRITER
PRUPOSE:	Extractions from common blocks to be written in the dynamic portion of the interactive display
NAME:	<u>IABRT (13)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of ABORT MISSION

NAME:	<u>IACC (10)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of ACCELERATE

NAME:	<u>IACTUL (6)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of ACTUAL

NAME:	<u>IALT (14)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of ALTITUDE (AGL)

NAME:	<u>IBOMB (12)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of BOMBING MODE

NAME:	<u>ICLB (5)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of CLIMB

NAME:	<u>IDEC (10)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of DECELERATE

NAME:	<u>IDEG (15)</u>
TYPE:	Local variable
USE:	Subroutine FLYAC
PURPOSE:	ASCII equivalent of ENTER DEGREES> Prompt for pilot input

NAME:	<u>IDEGS</u>
TYPE:	Local variable
USE:	Subroutine FLYAC, TURN
PURPOSE:	Heading change parameter

NAME:	<u>IDELAY (24)</u>
TYPE:	Local variable
USE:	Subroutine FLYAC
PURPOSE:	ASCII equivalent of ENTER DELAY MULTIPLIER

NAME:	<u>IDESC (7)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of DESCEND

NAME:	IDIREC
TYPE:	Local variable
USE:	Subroutine TURN
PURPOSE:	Input paramenter, indicates direction of turn
NAME:	IFEET
TYPE:	Local variable
USE:	Subroutine FLYAC, DESCND, CLIMB
PURPOSE:	Altitude change parameter
NAME:	IFIRST
TYPE:	Local variable
USE:	Subroutine WRITER
PURPOSE:	Loop parameter
NAME:	IFLIT (11)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of FLIGHT PATH
NAME:	IFLOW (19)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of FUEL FLOW
NAME:	IFREQ(17)
TYPE:	Local variable
USE:	Subroutine JAMMER
PURPOSE:	ASCII equivalent of ENTER FREQUENCY>
NAME:	IFUEL (14)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of FUEL REMAINING
NAME:	IHEAD (12)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of TRUE HEADING.
NAME:	ILAST
TYPE:	Local variable
USE:	Subroutine WRITER
PURPOSE:	Loop parameter
NAME:	ILMARG
TYPE:	Local constant
USE:	Subroutine DRLINE
PURPOSE:	Left edge of the menu

NAME:	<u>INCTIM</u>
TYPE:	Common variable (TSTEP)
USE:	Subroutine INPUT, GINITL, FLYAC
PURPOSE:	Time increment that the simulation "runs" between prompts for input

NAME:	<u>INDEX</u>
TYPE:	Common variable (II)
USE:	Subroutine CTRLC,
PURPOSE:	Flag to indicate that an interrupt has occured

NAME:	<u>INFO1</u>
TYPE:	Subroutine name
PURPOSE:	Prints information about the aircraft flight characteristics
CALLS:	CHRSIZ, NEWLIN, ANMODE, HDCOPY, NEWPAG
CALLED BY:	PRELIM

NAME:	<u>INFO2</u>
TYPE:	Subroutine name
PURPOSE:	Prints information about aircraft configuration capability
CALLS:	CHRSIZ, NEWLIN, ANMODE, NEWPAG
CALLED BY:	PRELIM

NAME:	<u>INFO3</u>
TYPE:	Subroutine name
PURPOSE:	Prints information about the simulation graphics displays and control characters
CALLS:	CHRSIZ, NEWLIN, ANMODE, HELP
CALLED BY:	PRELIM

NAME:	<u>INPUT</u>
TYPE:	Subroutine name
PURPOSE:	Interactive input routine to configure the aircraft prior to running the simulation
CALLS:	CHRSIZ, NEWPAG, ANMODE
CALLED BY:	PRELIM

NAME:	<u>INRNG</u>
TYPE:	Local variable
USE:	Subroutine RWRPLT
PURPOSE:	Counter for number of sites within radar range

NAME:	<u>INVAL (15)</u>
TYPE:	Local variable
USE:	Subroutine JAMMER
PURPOSE:	ASCII equivalent of INVALID INPUT

NAME:	<u>IPOS (8)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of POSITION

NAME:	IPREP (10)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of PREPLANNED
NAME:	IPRPLN (10)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of PREPLANNED
NAME:	IPWR (13)
TYPE:	Local variable
USE:	Subroutine JAMMER
PURPOSE:	ASCII equivalent of ENTER POWER>
NAME:	IRADR (30)
TYPE:	Local variable
USE:	Subroutine RWR
PURPOSE:	ASCII equivalent of RADAR WARNING RECEIVER DISPLAY
NAME:	IRBMB (10)
TYPE:	Local variable
USE:	Subroutine WEPWDS
PURPOSE:	ASCII equivalent of IRON BOMBS
NAME:	IREAL (20)
TYPE:	Local variable
USE:	Subroutine FLYAC
PURPOSE:	ASCII equivalent of ENTER A POS REAL #
NAME:	IRESM (6)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of RESUME
NAME:	IRFMIS
TYPE:	Common variable (WEAPNS)
USE:	Subroutine GINITL, INPUT
PURPOSE:	Number of rf missiles available
NAME:	IRMARG
TYPE:	Local constant
USE:	Subroutine DRLINE
PURPOSE:	Right edge of the menu
NAME:	IRMIS
TYPE:	Common variable (WEAPNS)
USE:	Subroutine GINITL, INPUT
PURPOSE:	Number of IR missiles available

NAME:	IRMISS (11)
TYPE:	Local variable
USE:	Subroutine WEPWDS
PURPOSE:	ASCII equivalent of IR missiles
NAME:	IRNBMB
TYPE:	Common variable (WEAPNS)
USE:	Subroutine GINITL, INPUT
PURPOSE:	Number of "iron bombs" available
NAME:	IRTCOL
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	X-coordinate in absolute units of the menu key words
NAME:	ISEC (14)
TYPE:	Local variable
USE:	Subroutine JAMMER
PURPOSE:	ASCII equivalent of ENTER SECTOR>
NAME:	ISECTR (5)
TYPE:	Common variable (JAMM)
USE:	Subroutine GINITL, INPUT, ECMVAL, JAMMER
PURPOSE:	Sector respective jammer is jamming
NAME:	ISMBMB
TYPE:	Common variable (WEAPNS)
USE:	Subroutine GINITL, INPUT
PURPOSE:	Number of "smart bombs" available
NAME:	ISPEED (12)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of GROUND SPEED
NAME:	ISTEP
TYPE:	Local constant
USE:	Subroutine CIRCLE
PURPOSE:	Roundness factor for the circle drawing algorithm.
NAME:	ITIME (12)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of ELAPSED TIME
NAME:	ITMSTP (16)
TYPE:	Local variable
USE:	Subroutine FLYAC
PURPOSE:	ASCII equivalent of ENTER TIMESTEP>

NAME:	ITRNL (9)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of TURN LEFT

NAME:	ITRNR (10)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of TURN RIGHT

NAME:	IY
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	Loop control used in computing which lines to write MAX in the menu

NAME:	IYVAL
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	Y-coordinate in absolute units of the incremental lines on which to write MAX in the the menu

NAME:	JALT (12)
TYPE:	Local variable
USE:	Subroutine FLYAC
PURPOSE:	ASCII equivalent of ENTER FEET> Prompt for pilot input

NAME:	JAMM
TYPE:	Common block name
USE:	Subroutine GINITL, INPUT, ECMWDS, ECMVAL, JAMMER
PURPOSE:	Data pertaining to jammers

NAME:	JAMMER
TYPE:	Subroutine name
PURPOSE:	Responds to pilot menu pick of JAMMER
CALLS:	BOXER, REFRSH
CALLED BY:	FLYAC

NAME:	JAMNUM (16)
TYPE:	Local variable
USE:	Subroutine JAMMER
PURPOSE:	ASCII equivalent of ENTER JAMMER #>

NAME:	JAMR (7)
TYPE:	Local variable
USE:	Subroutine ECMWDS
PURPOSE:	ASCII equivalent of JAMMER_

NAME:	JAMR (8)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of ECM MODE
NAME:	JMODE
TYPE:	Local variable
USE:	Subroutine JAMMER, FLYAC
PURPOSE:	Whether normal or max jammer mode was picked
NAME:	JX
TYPE:	Local variable
USE:	Subroutine FLYAC, JAMMER
PURPOSE:	X-coordinate at which to print prompt
NAME:	JY
TYPE:	Local variable
USE:	Subroutine FLYAC, JAMMER
PURPOSE:	Y-coordinate at which to print prompt
NAME:	KBOOM (6)
TYPE:	Local variable
USE:	Subroutine HITGND
PURPOSE:	ASCII equivalent of KABOOM
NAME:	KCHAFF (10)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of CHAFF LEFT
NAME:	KCHAR
TYPE:	Local variable
USE:	Subroutine PLTSIT
PURPOSE:	ASCII equivalent of the symbol to be plotted
NAME:	KDIFF
TYPE:	Local variable
USE:	Subroutine ABORT
PURPOSE:	Amount of change required to transition from present to abort parameter
NAME:	KECM (18)
TYPE:	Local variable
USE:	Subroutine ECMWDS
PURPOSE:	ASCII equivalent of CURRENT ECM STATUS
NAME:	KFLARE (11)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of FLARES LEFT

NAME: KFREQ (9)
TYPE: Local variable
USE: Subroutine ECMWDS
PURPOSE: ASCII equivalent of FREQ BAND

NAME: KHOUR
TYPE: Common variable (CLOCK)
USE: Subroutine GINITL, KPTIM, WRITER
PURPOSE: Simulated hours into the mission

NAME: KNOT (13)
TYPE: Local variable
USE: Subroutine FLYAC
PURPOSE: ASCII equivalent of ENTER KNOTS>
Prompt for pilot input

NAME: KNOTS
TYPE: Local variable
USE: Subroutine FLYAC, DECEL, ACCEL
PURPOSE: Speed change parameter

NAME: KOUT
TYPE: Local variable
USE: Subroutine WRITER, ECMVAL, WEPVAL, RUNSIM, PUTTIM
PURPOSE: Character equivalent of a number

NAME: KPK (4)
TYPE: Local variable
USE: Subroutine WEPWDS
PURPOSE: ASCII equivalent of _PK_

NAME: KPOWR (5)
TYPE: Local variable
USE: Subroutine ECMWDS
PURPOSE: ASCII equivalent of POWER

NAME: KPROM1 (15)
TYPE: Local variable
USE: Subroutine FLYAC, GSETUP
PURPOSE: ASCII equivalent of MAKE INPUTS NOW
Prompt for pilot input

NAME: KPROM2 (22)
TYPE: Local variable
USE: Subroutine FLYAC, GSETUP
PURPOSE: ASCII equivalent of TYPE "c" WHEN COMPLETE

NAME:	<u>KPTIM</u>
TYPE:	Subroutine name
USE:	Keeps time in hours, minutes, seconds
CALLS:	None
CALLED BY:	RUNSIM
NAME:	<u>KRFMIS (11)</u>
TYPE:	Local variable
USE:	Subroutine WEPWDS
PURPOSE:	ASCII equivalent of RF MISSILES
NAME:	<u>KSECND</u>
TYPE:	Common variable (CLOCK)
USE:	Subroutine GINITL, KPTIM, WRITER
PURPOSE:	Simulated seconds into the mission
NAME:	<u>KSECT (6)</u>
TYPE:	Local variable
USE:	Subroutine ECMWDS
PURPOSE:	ASCII equivalent of SECTOR
NAME:	<u>KSMBMB (11)</u>
TYPE:	Local variable
USE:	Subroutine WEPWDS
PURPOSE:	ASCII equivalent of SMART BOMBS
NAME:	<u>KTGT (13)</u>
TYPE:	Local variable
USE:	Subroutine WEAPON
PURPOSE:	ASCII equivalent of SELECT TARGET
NAME:	<u>KVIS (14)</u>
TYPE:	Local variable
USE:	Subroutine GETTGT
PURPOSE:	ASCII euqivalent of VISUAL DISPLAY
NAME:	<u>KWEP (5)</u>
TYPE:	Local variable
USE:	Subroutine WEPWDS
PURPOSE:	ASCII equivalent of CURRENT WEAPON STATUS
NAME:	<u>LCOL</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	X-coordinate in absolute units of the aircraft status key words
NAME:	<u>LEFT</u>
TYPE:	Local constant
USE:	Subroutine TURN, FLYAC
PURPOSE:	Indicates direction of turn

NAME:	LEFT (6)
TYPE:	Local variable
USE:	Subroutine WEPWDS
PURPOSE:	ASCII equivalent of #LEFT
NAME:	LFTCOL
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	X-coordinate in absolute units of the menu key words
NAME:	LFTING
TYPE:	Common variable (STATUS)
USE:	Subroutine GINITL, FLYAC, STATVL
PURPOSE:	TRUE when aircraft is turning left
NAME:	LIMITS
TYPE:	Common block name
USE:	Subroutine GINITL, STATVL
PURPOSE:	Aircraft flight capabilities
NAME:	LLX
TYPE:	Local variable
USE:	Subroutine BOXER
PURPOSE:	X-coordinate in absolute units of the lower left corner of the desired box
NAME:	LLY
TYPE:	Local variable
USE:	Subroutine BOXER
PURPOSE:	Y-coordinate in absolute units of the lower left corner of the desired box
NAME:	LOCATN
TYPE:	Common block name
USE:	Subroutine GINITL, STATVL
PURPOSE:	Position of the aircraft
NAME:	LVLOFF
TYPE:	Subroutine name
PURPOSE:	Resets status flag and fuel use rate to simulate the aircraft leveling off from an altitude change
CALLS:	RSETFF
CALLED BY:	ABORT, CLIMB, DESCND, FLYAC, HITGND, STATVL
NAME:	MAX (3)
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII equivalent of MAX

NAME:	<u>MAXACC</u>
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC
PURPOSE:	Aircraft maximum rate of acceleration.

NAME:	<u>MAXDN</u>
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft maximum descent rate

NAME:	<u>MAXSLO</u>
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft maximum rate of deceleration

NAME:	<u>MAXTRN</u>
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft maximum rate of turn

NAME:	<u>MAXUP</u>
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft maximum rate of climb

NAME:	<u>MEM</u>
TYPE:	Common block name
USE:	Subroutine REBOX, GINITL, ACCEL, CLIMB, DECEL, DESCND, LVLOFF, ROLOUT, SETSPD, TURN
PURPOSE:	Things to remember when refreshing the screen

NAME:	<u>MEMBOX (6,2)</u>
TYPE:	Common variable (MEM)
USE:	Subroutine REBOX, GINITL, ACCEL, CLIMB, DECEL, DESCND, LVLOFF, ROLOUT, SETSPD, TURN
PURPOSE:	Flags as to which menu box to highlight

NAME:	<u>MINIT</u>
TYPE:	Common variable (CLOCK)
USE:	Subroutine GINITL, KPTIM, WRITER
PURPOSE:	Simulated minutes into the mission

NAME:	<u>MISSN (22)</u>
TYPE:	Local variable
USE:	Subroutine WORDS
PURPOSE:	ASCII euqivalent of CURRENT MISSION STATUS

NAME:	<u>MISTIM</u>
TYPE:	Common variable (SECS)
USE:	Subroutine GINITL, STATVL, WRITER, PROCES
PURPOSE:	Elapsed mission time

NAME: **MOVABS**
TYPE: TEKTRONIX subroutine
PURPOSE: See TEKTRONIX documentation

NAME: **MOVEA**
TYPE: TEKTRONIX subroutine
PORPOSE: See TEKTRONIX documentation

NAME: **NDTIM**
TYPE: Common variable (SECS)
USE: Subroutine PROCES, GINITL
PURPOSE: Max length of simulation

[illegible]: [illegible]
[illegible]
[illegible]: [illegible]
[illegible]: [illegible] report

[illegible]: **NEWALT**
[illegible]: [illegible] variable (ACSTAT)
USE: Subroutine CLIMB, DESCND, GINITL
PURPOSE: Altitude to which the aircraft is transitioning

NAME: **NEWBEE**
TYPE: Local variable
USE: Subroutine REDY, AADRIVER
PURPOSE: TRUE when user requests operation assistance

NAME: **NEWHDG**
TYPE: Common variable (ACSTAT)
USE: Subroutine TURN, ROLOUT, GINITL
PURPOSE: Heading to which the aircraft is transitioning

NAME: **NEWLIN**
TYPE: TEKTRONIX subroutine
PURPOSE: See TEKTRONIX documentation

NAME: **NEWPAG**
TYPE: TEKTRONIX subroutine
PURPOSE: See TEKTRONIX documentation

NAME: **NEWSPD**
TYPE: Common variable (ACSTAT)
USE: Subroutine ACCEL, DECEL, SETSPD, GINITL
PURPPOSE: Speed to which the aircraft is transitioning

NAME: **NONE**
TYPE: Local variable
USE: Subroutine PIKSIT
PURPOSE: ASCII equivalent of NO SITES IN RANGE

NAME:	NORMDN
TYPE:	Common variable (ACPRAM)
USE:	Subroutie FLYAC, GINITL
PURPOSE:	Aircraft normal descent rate
NAME:	NORMUP
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft normal rate of climb
NAME:	NRMACC
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft normal rate of acceleration
NAME:	NRMSLO
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft normal rate of deceleration
NAME:	NRMTRN
TYPE:	Common variable (ACPRAM)
USE:	Subroutine FLYAC, GINITL
PURPOSE:	Aircraft normal rate of turn
NAME:	NTRGET
TYPE:	Common variable
USE:	Subroutine GETTGT
PURPOSE:	Site number of target selected by "pilot"
NAME:	NUMCHF
TYPE:	Common variable (WEAPNS)
USE:	Subroutine CHAFF, GINITL, INPUT
PURPOSE:	Number of pods of chaff available
NAME:	NUMFLR
TYPE:	Common variable (WEAPNS)
USE:	Subroutine FLARE, GINITL, INPUT
PURPOSE:	Number of flares available
NAME:	NUMJMR
TYPE:	Common variable (JAMM)
USE:	Subroutine GINITL, INPUT, ECMWDS
PURPOSE:	Number of jammers on the aircraft
NAME:	NUMPOS
TYPE:	Common variable (VISBLE)
USE:	Subroutine PIKSIT, PLTSIT
PURPOSE:	Number of sites which are possible to bomb

NAME:	PI
TYPE:	Local constant
USE:	Subroutine CIRCLE
PURPOSE:	Standard geometric value 3.141592

NAME:	PIKSIT
TYPE:	Subroutine name
PURPOSE:	Gets user pick of targets available
CALLS:	VCURSR, NEWPAG, ANMODE, HELP, REFRSH, PLTSIT, CHRSIZ, TRANGL, SWINDO, RROTAT
CALLED BY:	GETTGT

NAME:	PLOTIT
TYPE:	Local variable
USE:	Subroutine RWR PLOT
PURPOSE:	Signals if site has any live components to avoid plotting dead sites

NAME:	PLTSIT
TYPE:	Subroutine name
PURPOSE:	Plots all threat sites within visual range of aircraft
CALLS:	CHRSIZ, MOVABS, ANSTR, ANMODE, DWINDO, CIRCLE, SWINDO, POINTA, RROTAT, MOVEA, POINTR, ANCHO
CALLED BY:	PIKSIT, GETTGT

NAME:	POWER (5)
TYPE:	Common variable (JAMM)
USE:	Subroutine GINITL, INPUT, ECMVAL, JAMMER
PURPOSE:	Power setting of respective jammer

NAME:	POWER1
TYPE:	Local variable
USE:	Subroutine STRNTH, RWRPLT
PURPOSE:	Power of radar signal received at aircraft

NAME:	PRELIM
TYPE:	Subroutine name
PURPOSE:	Driver for all the preliminary tasks
CALLS:	INITT, TERM, WELCUM, GINITL, REDY, NEWPAG, INFO1, INFO2, INFO3, SETDEV, GSETUP, WAIT
CALLED BY:	AADRIVER

NAME:	PROCES
TYPE:	Subroutine name
PURPOSE:	Handles interupts and drives the interactive simulation
CALLS:	ENABLE, FLYAC, RUNSIM, REFRSH
CALLED BY:	AADRIVER

NAME:	<u>PUTTIM</u>
TYPE:	Subroutine name
PURPOSE:	Prints mission time in HR:MIN:SEC format
CALLS:	MOVABS, STRNUM, ANCHO, ANMODE
CALLED BY:	WRITER
NAME:	<u>PWRMAX</u>
TYPE:	Local variable
USE:	Subroutine RWRPLT
PURPOSE:	Maximum radar signal power being received at the aircraft
NAME:	<u>RADHDG</u>
TYPE:	Local variable
USE:	Subroutine PIKSIT, STATVL
PURPOSE:	Radian equivalent of aircraft heading
NAME:	<u>RADIUS</u>
TYPE:	Local variable
USE:	Subroutine CIRCLE
PURPOSE:	Input parameter/radius of the circle
NAME:	<u>RATE</u>
TYPE:	Local variable
USE:	Subroutine ABORT
PURPOSE:	Real equivalent of rate of change for required maneuvering
NAME:	<u>RDRRNG</u>
TYPE:	Local variable
USE:	Subroutine RWRPLT
PURPOSE:	"Line of sight" of radar
NAME:	<u>REBOX</u>
TYPE:	Subroutine name
PURPOSE:	Rehighlights menu boxes for which maneuver is still active after refresh
CALLS:	BOXER
CALLED BY:	REFRSH
NAME:	<u>REDY</u>
TYPE:	Subroutine name
PURPOSE:	Tells user file input complete and determines whether to print user operation assistance routines
CALLS:	CHRSIZ, ANMODE, NEWPAG
CALLED BY:	PRELIM
NAME:	<u>REFRSH</u>
TYPE:	Subroutine name
PURPOSE:	Controls the redrawing of the graphics display
CALLS:	NEWPAG, DISPLA, WRITER, REBOX
CALLED BY:	FLYAC, GSETUP. HITGND, PIKSIT, SHAKE, STALL, WEAPON, JAMMER

NAME:	RELATE
TYPE:	Subroutine name
PURPOSE:	Interfaces the interactive aircraft parameters with AADEM vehicle #1
CALLS:	None
CALLED BY:	RUNSIM

NAME:	RESUME
TYPE:	Subroutine name
PURPOSE:	Responds to pilot menu pick of RESUME INITIAL FLIGHT PATH
CALLS:	BOXER
CALLED BY:	FLYAC

NAME:	RFMSPK
TYPE:	Common variable (WEPPKS)
USE:	Subroutine GINITL, INPUT, WEAPON
PURPOSE:	Probability of kill for an RF MISSILE

NAME:	RLOSS
TYPE:	Local variable
USE:	Subroutine STRNTH
PURPOSE:	Loss factor for radar signal

NAME:	ROLOUT
TYPE:	Subroutine name
PURPOSE:	Resets status flag to indicate constant heading and resets the fuel use rate accordingly
CALLS:	RSETFF
CALLED BY:	ABORT, FLYAC, STATVL, TURN

NAME:	RROTAT
TYPE:	TEKTRONIX subroutine
PURPOSE:	See TEKTRONIX documentation

NAME:	RSETFF
TYPE:	Subroutine name
PURPOSE:	Adjusts aircraft fuel use rate at completion of a flight maneuver
CALLS:	NONE
CALLED BY:	ROLOUT, LVLOFF, SETSPD

NAME:	RTING
TYPE:	Common variable (STATUS)
USE:	Subroutine GINITL, FLYAC, STATVL
PURPOSE:	TRUE when aircraft is turning right

NAME:	RUNSIM
TYPE	Subroutine name
PURPOSE:	Updates all parameters of the simulation to reflect the time increment fed as an input argument then refreshes the screen to feed this information to the user
CALLS:	STATVL, VIEWS, RADPAR, RADSIG, JAMSIG, JTSCOM, BATTLE2, KPTIM
CALLED BY:	PROCES

NAME:	RWR
TYPE:	Subroutine name
PURPOSE:	Draws the RWR
CALLS:	MOVABS, ANSTR, SWINDO, CIRCLE, TRANGL
CALLED BY:	DISPLA

NAME:	RWRPLT
TYPE:	Subroutine name
PURPOSE:	Plots the sites for the RWR display
CALLS:	CHRSIZ, MOVABS, ANMODE, DWINDO, SWINDO RROTAT, STRNTH, MOVEA, POINTR
CALLED BY:	RWR

NAME:	SBMBPK
TYPE:	Common variable (WEPPKS)
USE:	Subroutine GINITL, INPUT, WEAPON
PURPOSE:	Probability of kill for a smart bomb

NAME:	SECS
TYPE:	Common block name
USE:	Subroutine GINITL, STATVL, WRITER PROCES
PURPOSE:	Elapsed mission time, total time allowed

NAME:	SETDEV
TYPE:	Subroutine name
PURPOSE:	Establish cursor home position
CALLS:	None
CALLED BY:	PRELIM

NAME:	SETFF
TYPE:	Subroutine name
PURPOSE:	Adjusts aircraft fuel use rate to begin a flight maneuver
CALLS:	None
CALLED BY:	ACCEL, CLIMB, DECEL, DESCND, TURN, WEAPON

NAME:	SETSPD
TYPE:	Subroutine name
PURPOSE:	Resets status flag and fuel use rate to simulate the aircraft stabilizing its speed from an acceleration or deceleration
CALLS:	RSETFF
CALLED BY:	ABORT, ACCEL, DECEL, FLYAC, SHAKE, STALL, STATVL

NAME: SHAKE
TYPE: Subroutine name
PURPOSE: Handles functions if the pilot attempts to exceed the aircraft's maximum speed
CALLS: ANMODE, REFRSH, BOXER, WAIT, ANCHO, SETSPD
CALLED BY: STATVL

NAME: SINT
TYPE: Local variable
USE: Subroutine CIRCLE
PURPOSE: Sine of THETA

NAME: SLOADJ
TYPE: Common variable (FUELS)
USE: Subroutine DECEL, SETSPD, GINITL
PURPOSE: Fuel use rate adjustment for deceleration

NAME: SLOING
TYPE: Common variable (STATUS)
USE: Subroutine GINITL, FLYAC, STATVL
PURPOSE: TRUE when aircraft is decelerating

NAME: SLOMAX
TYPE: Common variable (FLPRAM)
USE: Subroutine DECEL
PURPOSE: Fuel use rate adjustment for a maximum rate deceleration

NAME: SLONRM
TYPE: Common variable (FLPRAM)
USE: Subroutine DECEL
PURPOSE: Fuel use rate adjustment for a normal rate deceleration

NAME: SLORAT
TYPE: Local variable
USE: Subroutine FLYAC, DECEL
PURPOSE: Deceleration rate parameter

NAME: SLOW
TYPE: Local variable
USE: Subroutine DECEL
PURPOSE: FLAG to SETSPD to indicate stabilize speed from deceleration

NAME: SPEED
TYPE: Common variable (ACSTAT)
USE: Subroutine ACCEL, DECEL, SETSPD, GINITL
PURPOSE: Current aircraft airspeed

NAME:	STALL
TYPE:	Subroutine name
PURPOSE:	Handles functions if the pilot attempts to fly slower than the aircraft's minimum speed
CALLS:	REFRSH, BOXER, ANMODE, WAIT, ANCHO, HITGND, SETSPD
CALLED BY:	STATVL

NAME:	STATUS
TYPE:	Common block name
USE:	Subroutine GINITL, FLYAC, ABORT, LVLOFF, SETSPD, ROLOUT
PURPOSE:	Flags which indicate the transient states of the aircraft

NAME:	STATVL
TYPE:	Subroutine name
PURPOSE:	Computes the current mission status values
CALLS:	LVOFF, HITGND, SHAKE, SETSPD, STALL, ROLOUT
CALLED BY:	RUNSIM

NAME:	STEPSZ
TYPE:	Local constant
USE:	Subroutine CIRCLE
PURPOSE:	Roundness factor for the circle drawing algorithm

NAME:	STLSPD
TYPE:	Common variable (LIMITS)
USE:	Subroutine GINITL, STATVL
PURPOSE:	Slowest speed the aircraft can attain

NAME:	STRNTH
TYPE:	Subroutine name
PURPOSE:	Determines the power of a radar signal at the aircraft
CALLS:	None
CALLED BY:	RWRPLT

NAME:	STRNUM
TYPE:	TEKTRONIX subroutine
PURPOSE:	See "AFWAL Auxiliary PLOT-10 Routines"

NAME:	STRTSM
TYPE:	Subroutine name
PURPOSE:	Alows user inputs, then starts time for simulation
CALLS:	FLYAC
CALLED BY:	AADRIVER

NAME:	SWINDO
TYPE:	TEKTRONIX subroutine
PURPOSE:	See TEKTRONIX documentation

NAME:	TERM
TYPE:	TEKTRONIX subroutine
PURPOSE:	See TEKTRONIX documentation

NAME:	THETA
TYPE:	Local variable
USE:	Subroutine CIRCLE
PURPOSE:	Arc of the circle

NAME:	TOPSPD
TYPE:	Common variable (LIMITS)
USE:	Subroutine GINITL, STATVL
PURPOSE:	Fastest speed the aircraft can attain

NAME:	TRANGL
TYPE:	Subroutine name
PURPOSE:	Draws a triangle around the present cursor position
CALLS:	MOVER, DRAWR
CALLED BY:	PIKSIT, RWR

NAME:	TRNADJ
TYPE:	Common variable (FUELS)
USE:	Subroutine TURN, ROLOUT, GINITL
PURPOSE:	Fuel use rate adjustment for turning

NAME:	TRNMAX
TYPE:	Common variable (FLPRAM)
USE:	Subroutine TURN, GINITL
PURPOSE:	Fuel use rate adjustment for a maximum rate turn

NAME:	TRNNRM
TYPE:	Common variable (FLPRAM)
USE:	Subroutine TURN, GINITL
PURPOSE:	Fuel use rate adjustment for a normal rate turn

NAME:	TRNRAT
TYPE:	Local variable
USE:	Subroutine FLYAC, TURN
PURPOSE:	Turn rate parameter

NAME:	TSTEP
TYPE:	Common block name
USE:	Subroutine INPUT, GINITL, PROCES, FLYAC
PURPOSE:	Simulation timing parameters

NAME:	TURN
TYPE:	Subroutine name
PURPOSE:	Simulates pilot actions to cause the aircraft to turn
CALLS:	SETFF, BOXER, ROLOUT
CALLED BY:	ABORT, FLYAC

NAME:	UPADJ
TYPE:	Common variable (FUELS)
USE:	Subroutine CLIMB, LVLOFF, GINITL
PURPOSE:	Fuel use rate adjustment for a climb
NAME:	UPING
TYPE:	Common variable (STATUS)
USE:	Subroutine GINITL, FLYAC, STATVL
PURPOSE:	TRUE when aircraft is climbing
NAME:	UPMAX
TYPE:	Local variable
USE:	Subroutine CLIMB
PURPOSE:	Fuel use rate adjustment for a maximum rate climb
NAME:	UPNORM
TYPE:	Common variable (FLPRAM)
USE:	Subroutine CLIMB
PURPOSE:	Fuel use rate adjustment for a normal rate climb
NAME:	VCURSR
TYPE:	TEKTRONIX subroutine
PURPOSE:	See TEKTRONIX documentation
NAME:	VISRNG
TYPE:	Local variable
USE:	Subroutine PLTSIT
PURPOSE:	Radius of visibility of pilot from altitude
NAME:	VISSIT (3,50)
TYPE:	COMMON variable (VISBLE)
USE:	Subroutine PLTSIT, PIKSIT
PURPOSE:	Data about visible sites
NAME:	WAIT
TYPE:	Subroutine name
PURPOSE:	Delays processing to synchronize simulated time with real time
CALLS:	SYS$SETIMR, SYS$WAITFR
CALLED BY:	HITGND, SHAKE, STALL, PRELIM
NAME:	WATMLT
TYPE:	Common variable (TSTEP)
USE:	Subroutine GINITL, RUNSIM
PURPOSE:	Multiplier for the WAIT function Accelerate or decelerate the "real time" aspect of the simulation

NAME:	WEAP
TYPE:	Common logical variable (DISP)
USE:	Subroutine REFRSH, DISPLA, WRITER, GINITL, FLYAC, GSETUP, SHAKE, STALL, WEAPON, PIKSIT, JAMMER
PURPOSE:	TRUE when the CURRENT WEAPON STATUS block and Visual Display are to be presented

NAME:	WEAPNS
TYPE:	Common block name
USE:	Subroutines CHAFF, FLARE
PURPOSE:	All common information relating to weapons

NAME:	WEAPON
TYPE:	Subroutine name
PURPOSE:	Responds to pilot menu pick of WEAPON
CALLS:	REFRSH, GETTGT, ANSTR, ANMODE, NEWLIN, SETFF, BOXER, TARGET
CALLED BY:	FLYAC

NAME:	WELCUM
TYPE:	Subroutine name
PURPOSE:	Prints introductory message to amuse user while files are input
CALLS:	CHRSIZ, ANMODE
CALLED BY:	PRELIM

NAME:	WEPPKS
TYPE:	Common block name
USE:	Subroutine GINITL, INPUT, WEAPON
PURPOSE:	Probability of kills for the weapons

NAME:	WEPVAL
TYPE:	Subroutine name
PURPOSE:	Writes the values for the CURRENT WEAPON STATUS block
CALLS:	CHRSIZ, MOVABS, STRNUM ANSTR
CALLED BY:	WRITER

NAME:	WEPWDS
TYPE:	Subroutine name
PURPOSE:	Writes the weapon information for the CURRENT WEAPON STATUS block
CALLS:	MOVABS, ANSTR
CALLED BY:	WORDS

NAME:	WORDS
TYPE:	Subroutine name
PURPOSE:	Lays out the key words for the static part of the output display
CALLS:	CHRSIZ, MOVABS, ANCHO, ANSTR, ECMWDS, WEPWDS
CALLED BY:	DISPLA

NAME:	WRITER
TYPE:	Subroutine name
PURPOSE:	Writes all dynamic values in the lower rectangles of the interactive display
CALLS:	CHRSIZ, STRNUM, MOVABS, ANSTR, ECMVAL, WEPVAL, PUTTIM
CALLED BY:	REFRSH

NAME:	XDIS
TYPE:	Local variable
USE:	Subroutine PIKSIT, PLTSIT, RWRPLT
PURPOSE:	X coordinate distance from aircraft to site or from cursor to site

NAME:	XPRIME
TYPE:	Local variable
USE:	Subroutine PIKSIT
PURPOSE:	Transformed x coordinate from VCURSR

NAME:	XPT
TYPE:	Local variable
USE:	Subroutine RWRPLT
PURPOSE:	X coordinate of point to be plotted

NAME:	XSEC
TYPE:	Local variable
USE:	Subroutine STRNTH
PURPOSE:	Radar cross section of aircraft

NAME:	YDIS
TYPE:	Local variable
USE:	Subroutine PIKSIT, PLTSIT, RWRPLT
PURPOSE:	Y coordinate distance from aircraft to site or from cursor to site

NAME:	YPRIME
TYPE:	Local variable
USE:	Subroutine PIKSIT
PURPOSE:	Transformed Y coordinate from VCURSR

NAME:	YPT
TYPE:	Local variable
USE:	Subroutine RWRPLT
PURPOSE:	Y coordinate of point to be plotted

APPENDIX IV

MIRAGE User's Guide

Man-in-loop
Interactive
Real-time
Aircraft mission simulation using
Graphics to display the
Environment

Foreword

The MIRAGE software is an interactive operation of the AADEM model used in the Avionics Laboratory at Wright Patterson, Air Force Base. It allows vehicle one of the AADEM model to be interactively "flown" through the pre-programmed defensive network. It is basically the simulation of a "wild weasel" type aircraft attempting to penetrate enemy defensive systems and destroy preplanned targets. MIRAGE allows for altering direction, speed, and altitude, deploying weapons, and using radar and radar jamming equipment. The user is graphically shown his current environment as available through the use of radar equipment, or by looking out the aircraft windows. All other information generally available to a pilot is available in the various displays.

Table of Contents

MIRAGE Users' Guide

Introduction

This guide describes all user actions and displays involved in the operation of the MIRAGE software. The two parts of this guide correspond to the two phases of execution: pre-simulation and simulation. It chronologically discusses the execution of the MIRAGE system.

During the initial, or pre-simulation phase, the user is shown orientation and information displays and is given the opportunity to configure the aircraft specifically for the given mission.

The second, or simulation phase is menu-driven and uses the crosshair capacity of the TEKTRONIX 4016 terminal to pick the menu items, and some keyboard inputs to further define user desires. This guide includes sample displays from the MIRAGE execution which are fully explained.

The user should be familiar with this guide before attempting to operate the simulation. Anyone familiar with this guide will find the interface fairly 'friendly', and before long will be "flying" like an ace.

Any items marked with an asterisk (*) in the second section, "Operating the Simulation" have not been completely implemented.

Pre-Simulation

Displays

After execution of the program is inititated, a welcome display will be presented on the screen. This display has two purposes. First, it lets you know that you have successfully accessed the model. Second, it gives you something to look at while the AADEM files are being read, and the program variables are being initialized. Be patient -- no action on your part is required to advance the program, and no action will expedite it either! When the MIRAGE system is ready to work for you, it will say exactly that.

All of your required actions will be prompted, so you need only follow directions. The software will ask if you need instructions on how to operate the simulation. If you respond other than yes, the program skips to the aircraft configuration routines.

If you respond yes, the first display presents all performance characteristics for the aircraft being simulated. It includes the maximum flying speed, stall recovery altitude requirements, and normal and maximum rates for each of the following parameters:

A) Fuel use

B) Turn

C) Acceleration

D) Deceleration

E) Climb

F) Descent

A photocopy of this information is automatically made (if the device is turned on) since that is quite a bit to remember. When you respond that you are ready to continue, the next display, which discusses ECM and weapon capability of the aircraft, is presented.

The simulated aircraft can be loaded beyond the capability of any real aircraft. It can carry up to five electro-magnetic jammers, any number of iron bombs, smart bombs, RF missiles, IR missiles, chaff pods, and flares. Also, for each of the weapon and ECM items, the probability of kill (PK) or PK degrade factor can be anything from zero to one. This is certainly an impressive weapon platform, but remember that the validity and usefulness of the mission results depend on your realistic selection of aircraft payload.

You will later be offered the option to change the built in timestep of twenty seconds which controls the time-advance of the simulation. Because the TEKTRONIX terminal must be completely cleared and redrawn anytime something in the display must be changed, the information on the screen can never be absolutely current. The clearing and redrawing of the screen takes slightly less than one second. These factors force you to make a compromise decision. If you select a small timestep of two or three seconds, in order to have fairly current data displayed, the screen will be flashing and drawing so often that you probably won't be able to read and absorb much usable information. On the other hand, while a longer timestep allows ample time to study the display, plenty can happen "behind the scenes" that you may not realize until it is too late to respond. The default timestep is twenty seconds. For reference, at typical cruising airspeeds, this will usually be between two and three miles of ground distance. Since the timestep can be changed anytime during the simulation phase, until you gain experience with the displays, I recommend you not change it until in the "run" mode of the simulation.

Your final option is to change the "delay multiplier." MIRAGE is designed to run approximately synchronized to clock time, as discussed above, by use of a delaying function. The real time aspect can be accelerated or decelerated by adjusting the delay multiplier, which has default value of 1. Setting the delay multiplier to .5 for example will accelerate the simulation in the sense that the delay between refreshes will be half the timestep, but all parameters will still be adjusted by the full timestep. Likewise, setting the multiplier to 2 will decelerate the simulation to approximately double real time. An example of use for a multiplier less than one is where you know you are a long way from the enemy sites and want to get to where the action is in a hurry. A large timestep coupled to a small multiplier will get you across the territory in a hurry. Conversely, if you are in the midst of the action, and want to watch things develop, your choice should be a small timestep and a large multiplier. The delay multiplier, like the timestep can be changed during MIRAGE execution, so until you gain experience with the normal operation of the model, I recommend you not change it.

Configuring the Aircraft

The next pre-simulation function is the actual configuring of the aircraft to your specifications. If you follow the prompts, you can't go wrong. (If you don't follow the prompts, the software will tell you how you erred, and ask you to re-enter your parameter.) There is a default value for the number of pieces of each type of equipment and the associated PK factor, as well as the timestep and delay multiplier, so if you desire the normal configuration, simply respond as such when asked.

The final pre-simulation presentation reviews the run time displays and the control characters that are used to operate the simulation. I won't go into detail about it here, since it is a condensation of the material found in the next section of this guide.

Operating the Simulation

Interpreting the Displays

The two displays available while operating the MIRAGE software correspond to an offensive and defensive attitude of the user. In both cases, the display is made up of five sections. (Refer to figure A-1 and figure A-2 as needed.) The upper left section of both displays is the interactive scratch pad. If the software requires an input, it will prompt you for it in this box. Your responses to the prompts will also be echoed here.

The lower left section of the display contains the CURRENT MISSION STATUS box. Here you will find the typical navigation and performance equipment readings available to a pilot. These are:

A) Elapsed time since the beginning of the simulation (does not include "timeouts" for user inputs as discussed below)

B) True heading (North=000, South=180, etc.)

C) Altitude in feet above ground level (AGL)

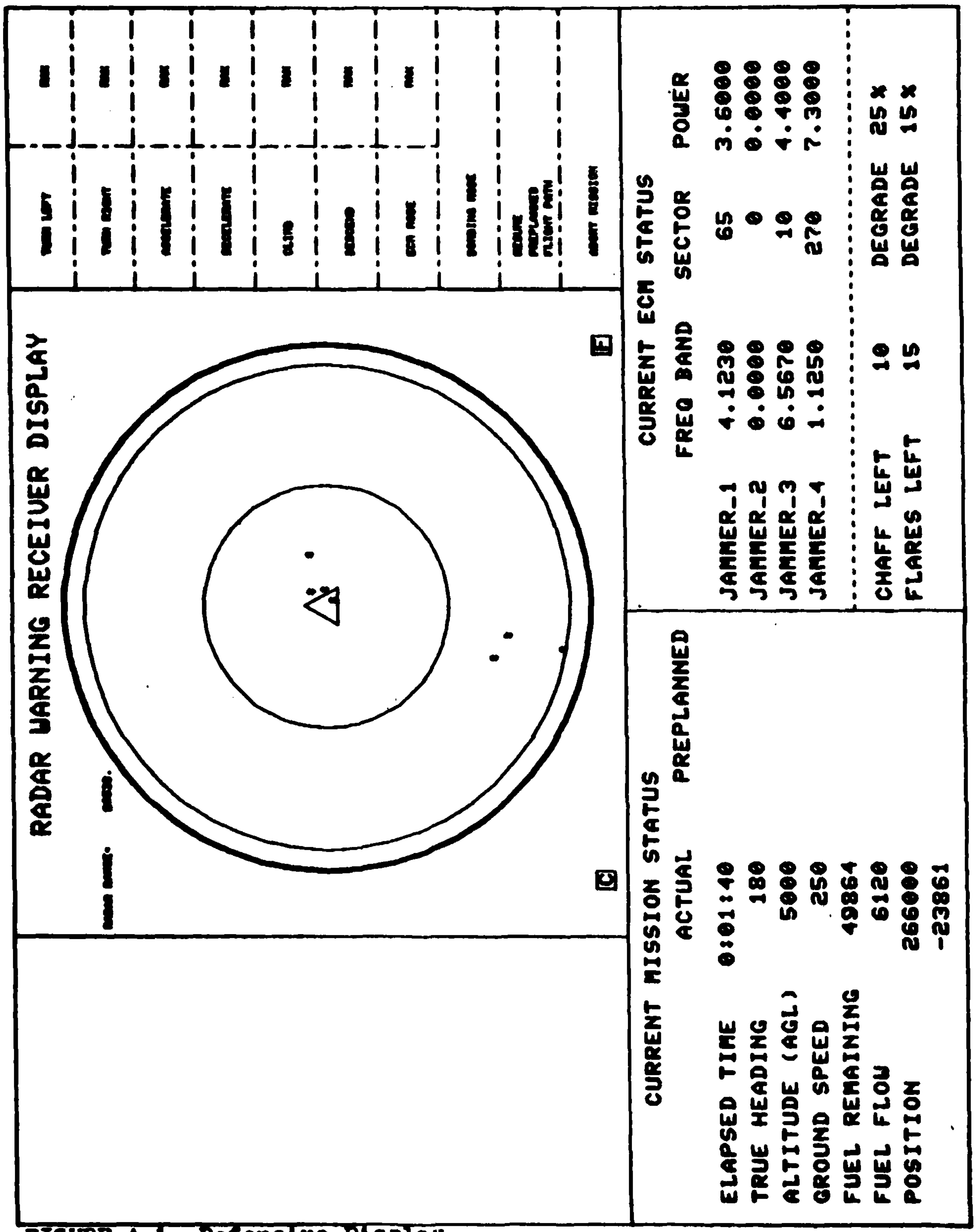
D) Ground speed in nautical miles (6,082.2 feet) per hour

E) Fuel remaining in pounds

F) Fuel flow (burn rate) in pounds per hour

G) Position (latitude and longitude*)

The second column in this box displays the preplanned mission parameters for the same elements discussed above.* This structure simulates the pilot's ability to compare his



CURRENT MISSION STATUS

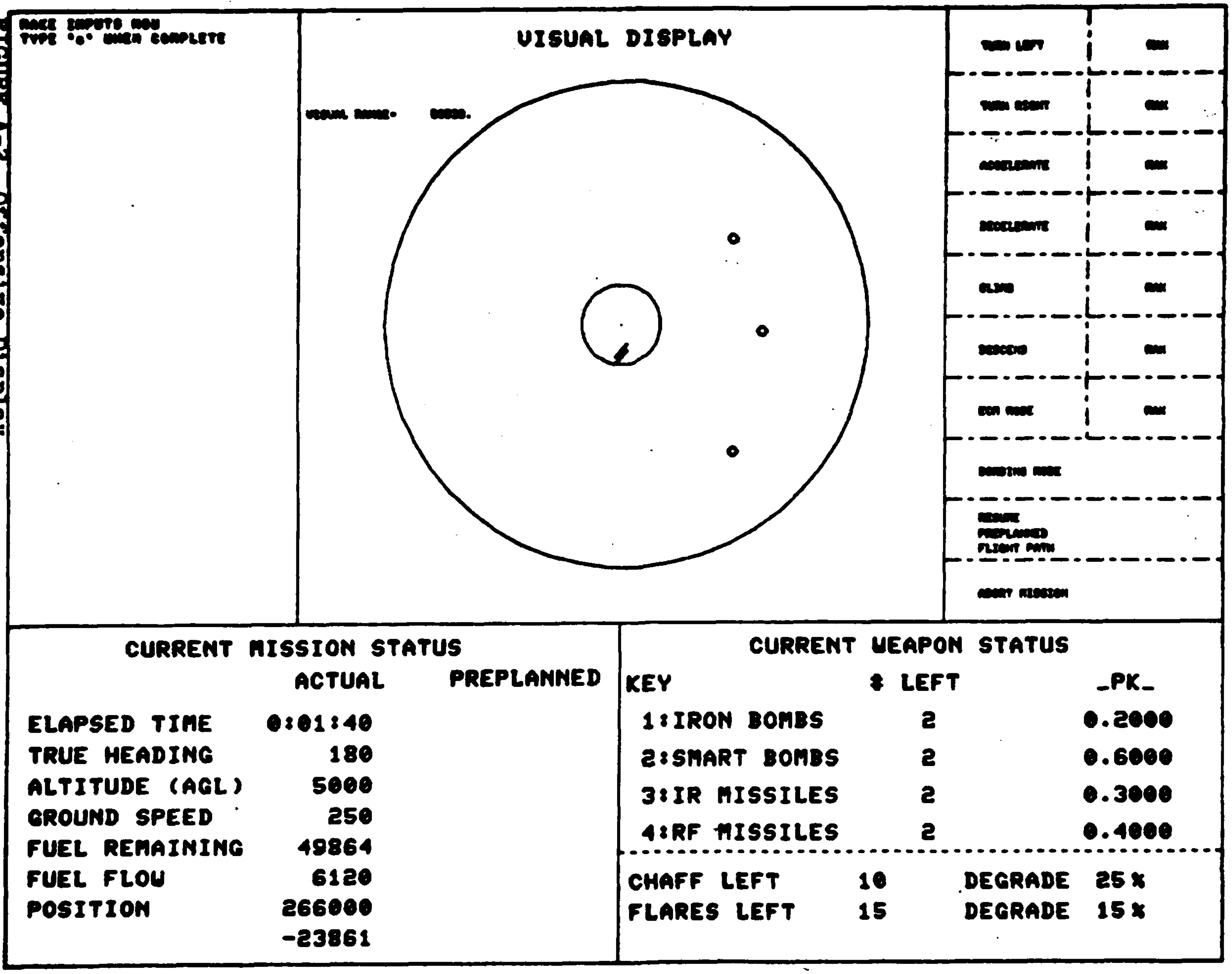
	ACTUAL	PREPLANNED
ELAPSED TIME	0:01:40	
TRUE HEADING	180	
ALTITUDE (AGL)	5000	
GROUND SPEED	250	
FUEL REMAINING	49864	
FUEL FLOW	6120	
POSITION	266000	
	-23861	

CURRENT ECM STATUS

	FREQ BAND	SECTOR	POWER
JAMMER_1	4.1230	65	3.6000
JAMMER_2	0.0000	0	0.0000
JAMMER_3	6.5670	10	4.4000
JAMMER_4	1.1250	270	7.3000

CHAFF LEFT	10	DEGRADE	25 x
FLARES LEFT	15	DEGRADE	15 x

FIGURE A-1 Defensive Display



```
MAKE INPUTS NOW
TYPE "G" WHEN COMPLETE
```

```
VISUAL DISPLAY

VISUAL RANGE=     50000.
```

```
TURN LEFT                MAX
TURN RIGHT               MAX
ACCELERATE               MAX
DECELERATE               MAX
CLIMB                    MAX
DESCEND                  MAX
ECM MODE                 MAX
BOMBING MODE
RESUME
PREPLANNED
FLIGHT PATH
ABORT MISSION
```

```
        CURRENT MISSION STATUS
                         ACTUAL     PREPLANNED

ELAPSED TIME            0:01:40
TRUE HEADING                180
ALTITUDE (AGL)             5000
GROUND SPEED                250
FUEL REMAINING            49864
FUEL FLOW                  6120
POSITION                 266000
                         -23861
```

```
                 CURRENT WEAPON STATUS
KEY                        # LEFT           _PK_
 1:IRON BOMBS                2            0.2000
 2:SMART BOMBS               2            0.6000
 3:IR MISSILES               2            0.3000
 4:RF MISSILES               2            0.4000
----------------------------------------------------
CHAFF LEFT           10        DEGRADE  25%
FLARES LEFT          15        DEGRADE  15%
```

FIGURE A-2 Offensive Display

navigation instruments to his charts and flight plan and correct accordingly.

The upper right section of the display is the interactive menu. It consists of ten items for selection, divided in some cases into two columns. With this menu, you control all aircraft maneuvers. The top six elements correspond to the pilot using the ailerons, elevators, rudder, and throttle(s) to fly the aircraft. The two columns represent _rate_ of movement in the selected direction. The left column represents _normal rate_, the right column _maximum rate_. Recall that from the pre-simulation display, if you had requested the information, you were furnished with the values for these rates for your aircraft. The ECM MODE box also has two columns. Picking this level will cause the displays to shift to the defensive mode which will be discussed later, and allow for adjusting the jammers, deploying chaff, or deploying flares. The MAX column sets the "max confuser" mode for the jamming equipment. Picking the BOMBING MODE box will cause the displays to shift to the offensive mode which will be discussed below, and allows for selecting targets and deploying weapons. Selecting the RESUME PREPLANNED FLIGHT PATH box will cause the aircraft to return to its preplanned route of flight.* Selecting the ABORT box causes the aircraft to maneuver, using normal rates, to its abort heading, altitude, and airspeed. This simulates the pilot's initial inputs to depart the enemy territory. It does not hamper the pilot's ability to make other inputs in any way.

The three sections described above are drawn every time the screen is refreshed. Which of the following two pairs of sections is also drawn depends on whether the last menu pick was ECM MODE or BOMBING MODE.

The Visual Display and CURRENT WEAPON STATUS box (see figure A-2) will be drawn whenever BOMBING MODE is picked, and every refresh cycle thereafter until the ECM MODE (or MAX) is selected. The weapon status box in the lower right section of the display, lists the available weapons on board by name with a corresponding "key", the number of that weapon type remaining, and the probability of kill for that weapon type. When the supply of a weapon type is exhausted, the name and other data are removed from the list. Similar information for chaff and flares is shown below the weapons, however chaff and flare information is also displayed with the ECM status which will be discussed shortly. When in the BOMBING or offensive mode, a visual display will appear in the upper center of the display. The upper left of this display shows the visual range in meters indicated by the outer circle. The display is oriented to the present aircraft heading, so that the top of the display is the front of the aircraft; consider it a top view of your environment with your aircraft in the center.

There are four symbols used on this display:

A) A "o" indicates that the object is too far away to distinguish

B) A "•" indicates a destroyed site

C) A "/" indicates a AAA site

D) A "^" indicates a SAM site

From higher altitudes, your visual range can exceed 100 miles, so it would be unrealistic to allow you to bomb anything in sight. However, it is equally unrealistic to force you to fly the simulator to the accuracy expected of our bomber pilots. To compromise, when visual range exceeds approximately seven miles, a smaller circle will appear on the visual display. This circle represents which of the displayed sites you will be allowed to bomb from your present position. You might want to think of it as the eyepiece of a bombsight. If only one circle is displayed, any sites visible may be bombed.

When the ECM MODE or its MAX box are picked, the CURRENT ECM STATUS and RADAR WARNING RECEIVER display are drawn. The ECM status box shows the identification character for each jammer, the code number for its frequency band, the center of the sector selected in degrees (i.e. 000 for North, 090 for East, etc.) and the code number for the power. Note that ^ displayed for the power indicates that the jammer is off.

The other defensive mode display is the RADAR WARNING RECEIVER (RWR). Similar to the Visual Display, in the upper left hand section the radar range is shown, This, like the visual range factor is the maximum possible from the given altitude. It does not consider terrain or other obstructions. Any site emitting a radar pulse which is received at the aircraft is displayed. Contrast this with the visual display which only shows threats. Headquarters, army field units, etc. may show up on the RWR if they are monitoring the aircraft with their radar, but unless there is SAM or AAA equipment at that location, they will not be on the visual display. On the RWR, all signals are plotted with the same symbol. However, the strongest signal received is plotted on the outer circle. Let me say that again: The strongest signal, possibly the closest site to the aircraft, is plotted farthest away from the triangle representing the aircraft in the center. This is representative of Radar Warning Receivers being used in actual Air Force aircraft today. It gives very good relative bearing (azimuth) information since aircraft heading is considered to be at the top, and it prevents the sites of importance (the strong signals) from hiding each other, which would happen if they were all plotted in the center of the display.

One final note. When the RWR display is shown, there are two "buttons" which are active. To the left is the "C" or chaff button, and to the right, the "F" button. These buttons, as well as how to interact with the rest of the simulation controls will be explained in the next section.

Interacting with the Simulation

The simulation is interupt driven, running in approximately real time, when it is running, and in a "time out" state while you are making inputs. Obviously, the crucial things to know are how to stop it so that you can interact, and how to start it again when your inputs are complete. To stop the simulation, enter a control c. That is, hold the 'CTRL' key down and type a "c". The simulation will stop, and the TEKTRONIX cursor will appear. (The TEKTRONIX cursor consists of a very fine line horizontally and vertically through the entire display.) If you don't see the cursor, rotate the white thumb wheels which are to the right of the keyboard until they appear. The menu item for a maneuver is "picked" by positioning the cursor intersection in the appropriate box and pressing the space bar. If the software needs more information, it prompts you for it in the interactive area. When all requirements for the maneuver are complete, the cursor will again appear. This allows you to make as many inputs as you desire before restarting the simulation. When you have entered all of the maneuvers you have in mind for the "timeout" period, simply enter a lower case "c" to continue. This is also how you initially start the simulation. That is all there is to it. First let's look at an example, and then some special cases.

You are cruising along at 15,000 feet, and decide that you would see a lot better from lower altitude, and at the same time hide from some of the enemy radar. First, you have to stop the simulation so that you can interact. Hold the "CTRL" key and enter a "c". The cursor should now be visible on the screen (it's not very bright). If you can't find it, rotate the thumb wheels a bit. Now that you have the cursor in sight, you must decide which rate of descent you want to make. Remember the left column is normal, the right column maximum rate in the menu. Now, rotate the thumb wheels to position the cursor on your menu choice and tap the space bar. You have now told the software to descend, but now you must specify how far. The program will prompt you with ENTER FEET>. Type the number of feet you wish to descend, and a "return" key to enter your choice. The menu should display a dashed box around your menu pick to show you that it understands your command, and the cursor will return to indicate that the computer is ready for the next command. Let me repeat an important point. When maneuvering the aircraft, the prompts are for the change desired not the destination. So it's knots to change, not final airspeed, degrees to turn, not new heading, etc.

The top six of the ten menu items work as outlined above. Pick with the cursor and space bar, then refine your command with keyboard inputs by responding to the prompts in the interactive area.

Selecting the ECM MODE or its Max box is slightly different from the above. First, the screen will refresh to display the defensive mode: RADAR WARNING RECEIVER and CURRENT ECM STATUS. You will then be prompted to enter the jammer identifier, and the power desired. If you enter zero for power, you have turned that jammer off. Otherwise you will be prompted to enter the frequency code and sector. The interpretation of the power and frequency codes is given in TABLE 1.*

Note: The defensive mode display will be presented for each refresh cycle until the BOMBING MODE box is selected.

Four items require only the cursor/space bar pick. These are the C button to deploy a chaff pod, the F button to deploy an infra-red flare, the ABORT MISSION box which maneuvers the aircraft to its preset abort profile, and the RESUME PREPLANED FLIGHT PATH* box, which manuevers the aircraft back onto its preplanned route. This leaves only the BOMBING MODE box to explain.

Selecting the BOMBING MODE box allows you to deploy <u>one</u> of your remaining weapons. This pick will cause the visual display to be updated and shown, and the cursor to again be presented. If a smaller circle is drawn on the visual display, this delimits the sites which can be bombed from your present position. If no sites are within range, MIRAGE will tell you so, and the cursor is again the menu pick cursor. If there are sites within range, you <u>must</u> pick one with the cursor to select as the target. Again, move the cursor to the site on the visual display and tap the space bar. You will now be prompted for a weapon.

Enter the key number of the weapon you want to use, or a zero if you would rather not attack, followed by the "return" key to enter your choice. The cursor will be ready to pick your next command.

Note: To deploy another weapon, you must re-select BOMBING MODE and proceed as above. You will not be allowed to pick a second target with the command cursor.

Note: The offensive mode display will be presented for each refresh cycle until the ECM mode or MAX box is selected.

By now you may be thinking, with all that going on I'm bound to make a mistake. It's not as bad as it sounds. Remember, you're in a timeout mode, so, with only a few exceptions you can recover from an errant pick. First, the exceptions: deploying chaff, flares, and weapons. When the hardware is launched it's gone; you can't call it back. The remaining maneuvers can be undone simply. Picking a change in direction, speed, or altitude will logically over-write a previously indicated change in direction, speed, or altitude respectively. You can certainly turn left and accelerate simultaneously, but you cannot climb and descend. In other words, if you enterred a "TURN LEFT 30 DEGREES", but you meant right, go back and pick "TURN RIGHT" and enter "30" when prompted for degrees. Only the last maneuver of each type (change in direction, speed, or altitude) will be executed. Maneuvers in the same direction do not combine, either. Ten ACCELERATE 20 KNOTS are the same as one. To cancel a maneuver altogether, repick it and enter zero for the amount. If you have had a particularly bad "timeout", and the menu or interactive scratch pad are confusing you, it is possible to clear the display and see your present status. This control character and the others will be described next.

Six special characters operate when in the interactive mode. To refresh the screen, and only show the highlighting boxes for the current flight maneuvers, enter a lower case "r". To change the timestep, that is, the period of simulated time which will elapse before the next automatic redrawing of the screen, enter a lower case "t". To change the delay multiplier, and alter the real time aspect of the simulation, enter a lower case "d". To signal that your inputs are complete, and you wish to continue the simulation, enter a lower case "c". To have a help message displayed, enter a lower case "h". To terminate the program altogether, and exit to the command mode of the computer, enter an upper case "T". Remember, these only work when you are in the interactive mode. To get there from the run mode, hold the CTRL key and type a "c". The control functions are summarized in Figure A3 below.

Control Character	Effect
^c	Stop the simulation/ allow for inputs
r	Refresh the screen
h	Display a help message
t	Change the timestep
d	Change delay multiplier
c	Continue simulation
T	Terminate the program

FIGURE A-3
Control Characters

NOTE: Entering a control c while the screen is being redrawn will cause a scrambled display. It does no harm, however. To get a usable display enter a lower case "r", and continue with your planned interaction.

With your new understanding of the displays as described above, and the commands used to interact with the program, you're now ready to run the program. I'm sure you will enjoy it!

VITA

Michael James Goci was born on 23 May, 1950, in Wyandotte, Michigan. He graduated from Cass Technical High School in 1968. He attended Oakland University at Rochester, Michigan and received a Bachelor of Arts Degree in Mathematics in April 1972.

In August, 1972, he entered the Air Force through Officer Training School. Upon receiving his commission in November, 1972, he was assigned to Laughlin Air Force Base, Texas for Undergraduate Pilot Training. Following graduation in February 1974, he was assigned to Webb Air Force Base, Texas as a T-37 Instructor Pilot in the Security Assistance Training Program. He trained students from Vietnam, Iran, several African nations, and various South American countries as well as one class of American students.

In December 1977, he was transferred to Pease Air Force Base, New Hampshire where he was an Aircraft Commander in the KC-135 and an instrument flight instructor. He completed the Master of Management Science Program from the University of Northern Colorado in March 1980.

In June 1981, he was assigned to the Air Force Institute of Technology to study toward a Master's Degree in computer science. He is also an active member of Tau Beta Pi.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/MA/82D-4	2. GOVT ACCESSION NO. AD-A124	3. RECIPIENT'S CATALOG NUMBER 667
4. TITLE (and Subtitle) AN INTERACTIVE BOMBING MISSION SIMULATION WITH COMPUTER GRAPHICS INTERFACE		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael J. Goci Capt		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Avionics Laboratory, Electronics Warfare Division (AFWAL/AAWA) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE December, 1982
		13. NUMBER OF PAGES 142
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR [illegible]

[illegible]

LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

~~Approved for public release; IAW AFR 190-17~~

~~FREDRIC C. LYNCH, Major, USAF~~
~~Director, Public~~ Affairs

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Graphics
Computer Simulation
Computer Modeling

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An interactive flight simulation with computer graphics interface was designed using top-down structured analysis techniques. The project converts a passive bombing mission simulation used in the Avionics Laboratory, Air Force Wright Aeronautical Laboratories at Wright-Patterson AFB, into an interactive, real-time, man-in-loop simulation. The design was documented using SofTech's Structured Analysis and Design

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Technique (SADT) then coded in FORTRAN. The graphics were implemented using TEKTRONIX PLOT-10 software and the system operates on a VAX-11/780 computer coupled through a TEKTRONIX 4016 terminal.